Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Classical finite-element solvers as differentiable graph programs on GPUs

Teeratorn Kadeethum

tkadeethum@gmail.com

Siemens Energy

Babak Badnava

Siemens Energy

Vignesh Athreya

Siemens Energy

Boyang Chen

Imperial College London

Claire Heaney

Imperial College London

Francesco Ballarin

Universita Cattolica del Sacro Cuore

Harpreet Sethi

Nvidia

Hongying Li

Nanyang Technological University

Christopher Pain

Imperial College London

Article

Keywords:

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11156288/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Classical finite-element solvers as differentiable graph programs on GPUs

Teeratorn Kadeethum[1,*], Babak Badnava[1], Vignesh Athreya[1], Boyang Chen[2], Claire E. Heaney[2,3,*], Francesco Ballarin[5,6], Harpreet Sethi[7], Hongying Li[8], Christopher C. Pain[2,3,4,*]

[1]Siemens Energy, AI Lab, Orlando, USA; [2]Department of Earth Science and Engineering, Applied Modelling and Computation Group, Imperial College London, SW7 2AZ, UK; [3]Centre for AI-Physics Modelling, Imperial-X, Imperial College London, W12 7SL, UK; [4]Data Science Institute, Data Assimilation Laboratory, Imperial College London, SW7 2AZ, UK; [5]Department of Mathematics and Physics, Università Cattolica del Sacro Cuore, via Garzetta 48, 25133 Brescia, Italy; [6]Department of Mathematics for Economic, Financial and Actuarial Sciences, Università Cattolica del Sacro Cuore, via Necchi 9, 20123 Milano, Italy; [7]NVIDIA, USA; [8]School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore

*Correspondence: tkadeethum@gmail.com, c.heaney@imperial.ac.uk, c.pain@imperial.ac.uk>

September 29, 2026

Abstract

Conjugate heat transfer in turbo-machinery, heat exchangers, and energy systems couples buoyant flow to the surrounding solid on complex geometries. The cost of simulating it is dominated by large sparse linear solves on CPUs. We recast a mixed finite element discretization on unstructured meshes, and its multigrid pressure solve, as sparse graph layers within an automatic differentiation framework. Nothing is trained: weights are entries of the assembled operator, and manufactured solutions recover theoretical convergence orders. Reverse-mode sensitivities match central finite differences; one gradient costs 5.6 forward solves with unrolling and 4.9 with implicit differentiation, nearly constant from one to 32 parameters. Implicit differentiation through every linear solve substantially reduces truncation-induced sensitivity error and makes memory independent of forward iteration count. Against three established solvers, no single code is fastest; the ranking depends on the regime, the size of the problem, and the accuracy demanded. With the discretization, mesh, time step, and tolerances held fixed, the same problem runs up to

181× faster per step on one GPU than on 32 CPU cores, and the pressure solve alone runs 18.7× faster. The graph representation depends on operator sparsity rather than the governing equations, providing a route to other assembled discretizations.

1 Introduction

Thermal design couples a moving fluid to the adjacent solid, each influencing the other's state. A practical solver must capture short-term dynamics and long-term transport while representing curved boundaries and solid–fluid interfaces, motivating a body-fitted unstructured mesh. We use natural convection and conjugate heat transfer (CHT) to demonstrate this: they fully couple heat exchange and transport and govern thermal management over a range of applications from electronics to engines and nuclear facilities.

Conjugate buoyant flow is traditionally modelled by coupling the Boussinesq Navier–Stokes equations to energy equations in the fluid and solid. Finite-volume and finite element discretizations commonly use segregated pressure-correction methods: the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) for steady calculations [1], and for transient work projection or PISO-type coupling with a time integrator, such as the semi-implicit third-order Runge–Kutta/Crank–Nicolson (RK3–CN) scheme used here. OpenFOAM provides established finite volume implementations [2], and FEniCSx supports flexible finite element formulations on body-fitted unstructured meshes [3, 4]. These workflows remain largely CPU-based, although GPU acceleration is advancing [5–7], driven by the much higher throughput that GPUs offer for suitably structured workloads.

Throughput is not the only consideration when choosing a solver: when written in an automatic-differentiation framework, solvers can also yield derivatives of discrete outputs with respect to physical and design parameters, for optimization, control, and uncertainty quantification [8–12]. Conventional solvers can also provide these derivatives, but usually through a custom adjoint. One way to obtain both throughput and differentiability is to express classical operators as fixed neural network layers. In Neural Physics [13], finite difference and finite element stencils on a Cartesian grid become convolution kernels with analytical, non-trainable weights, so the network applies the classical schemes' discrete operators. Solvers built this way have been demonstrated on fluid flow benchmarks [13, 14], shallow water equations [15], buoyancy-driven flows [11], neutron diffusion [16] and radiation transport [17]. Cartesian convolutions, however, do not naturally support body-fitted meshes or curved solid–fluid interfaces.

Recent work has extended GPU-native methods beyond Cartesian grids [18–21], complementing established GPU-capable finite- and spectral-element software and high-order element-operator frameworks [22–25]. Our solver instead retains a classical assembled fi-

nite element formulation, with FEniCSx as the assembly backend [3] and the mixed P_2–P_1 Taylor–Hood pair [26]. The Poisson equation representing the pressure system is solved with the conjugate gradient method, preconditioned with a smoothed-aggregation algebraic multigrid [27–33], and the distributed implementation draws on established coarse space constructions, including deflation within the conjugate gradient iteration [34], substructuring by constrained energy minimization [35], and spectral coarse spaces built from local generalized eigenproblems [36].

We express the assembled finite element and multigrid operators as fixed graph operations. Nodes are degrees of freedom (DOF), edges are nonzero matrix couplings, and weights are set by the discretization. Transfer operators, smoothers, and coarse corrections use the same representation, so the graph U-Net reproduces the algebraic-multigrid V-cycle within floating-point and solver tolerances. Nothing is trained; "graph neural network" (GNN) describes the execution structure, not a learned model. This follows the ideas in Neural Physics [13] and algorithm unrolling [37–39], differing from the learned graph simulators [40–42]. Each operator application is a weighted sum over a degree of freedom's neighbours (Equation (8)), depending only on the sparsity of the assembled operator, not the equations it discretizes, so the construction is not tied to the Boussinesq system demonstrated here. Our contribution is an end-to-end differentiable, GPU-native CHT solver on unstructured meshes.

We evaluate the approach with four solvers: a structured-grid convolutional neural network (CNN) based on the Neural Physics idea; the GNN solver proposed here with its multigrid pressure preconditioner as a graph U-Net (Figure 1); a reference FEniCSx P_2–P_1 solver [3]; and OpenFOAM v12 [2, 43]. We evaluate these solvers across five dimensionless Boussinesq configurations. All four are compared on the steady cavity. The configurations are: a differentially heated cavity in two and three dimensions, a turbulent Rayleigh–Bénard cell, a conjugate cavity enclosing a conducting solid, a branched conducting heat sink, and a single-cylinder engine. All four are tested in the steady cavity, where they converge to the same Nusselt number. The benchmark design and timing protocol are given in Section 4 and Section 4.2.

Every solver considered here, including ours, solves the discretized equations rather than a trained surrogate. Although our solver is written as a GNN, it preserves the underlying discrete method rather than approximating it. Manufactured solutions recover the theoretically predicted convergence orders in three dimensions, while the graph V-cycle reproduces a reference implementation to fp64 round-off (Table 1; Supplementary Note S1). The computation is end-to-end differentiable (Figure 2; Supplementary Note S2), and the implicit rule targets the sensitivity of the converged solution rather than the derivative of the truncated iteration. The implementation is distributed across eight H200 GPUs, reaching hundreds of millions of DOF on a conjugate heat sink and

engine geometry (Section 2.7); a two-level global coarse correction reduces the growth of the pressure iteration count with the number of subdomains (Extended Data Fig. 1). The distributed contribution is therefore the ability to sustain this capacity together with coarse global correction, rather than efficient strong scaling.

2 Results

Results are reported on the five configurations of Section 4: the differentially heated cavity in 2D and 3D, the turbulent Rayleigh–Bénard cell, the conjugate cavity, the branched heat sink and the engine cylinder. All four solvers are tested on the steady cavity, where they converge to the same Nusselt number.

2.1 Verification, grid convergence, and benchmark agreement

Table 1 collects the correctness evidence; only the validation block compares against external values. The 2D cavity is the differentially heated square cavity of de Vahl Davis [44] (air, $\mathrm{Pr} = 0.71$). At $\mathrm{Ra} = 10^6$ we reference Le Quéré [45]'s spectral $\mathrm{Nu} = 8.8252$ rather than de Vahl Davis's 1983 value: the 0.3% offset is a difference between the references, and all four solvers cluster on the spectral one, so the consensus coincides with the benchmark. The centreline velocity profiles also reproduce the benchmark, to 1.5% at h_{128}. The full benchmark record is Supplementary Note S3: profiles in Supplementary Figure S3 and per-solver Nusselt values in Supplementary Table S7.

Every entry in Extended Data Table 1 and Supplementary Table S5 meets the stopping test.

2.2 A verified gradient through the coupled solve

The solver is one automatic-differentiation graph, so sensitivities follow by reverse-mode differentiation without a hand-derived adjoint. The solid–fluid conductivity ratio k_r is carried as a differentiable scalar through the affine thermal operator. A central finite difference converges to the reverse-mode sensitivity at second order before round-off dominates, agreeing to 2.6×10^{-9} at best (Figure 2a) and to 10^{-9} to 2.7×10^{-6} over $k_r \in [0.25, 100]$ (Supplementary Note S2.1). Both computations evaluate the same fixed-iteration program, so this establishes only that reverse mode differentiates that program correctly.

Whether that derivative is the sensitivity of the discrete solution depends on how far the solve has converged. Differentiating a fixed number of Krylov sweeps reaches it only slowly, and a finite difference through the same program cannot detect the shortfall. We therefore carry the implicit rule through every linear solve of the coupled step, storing

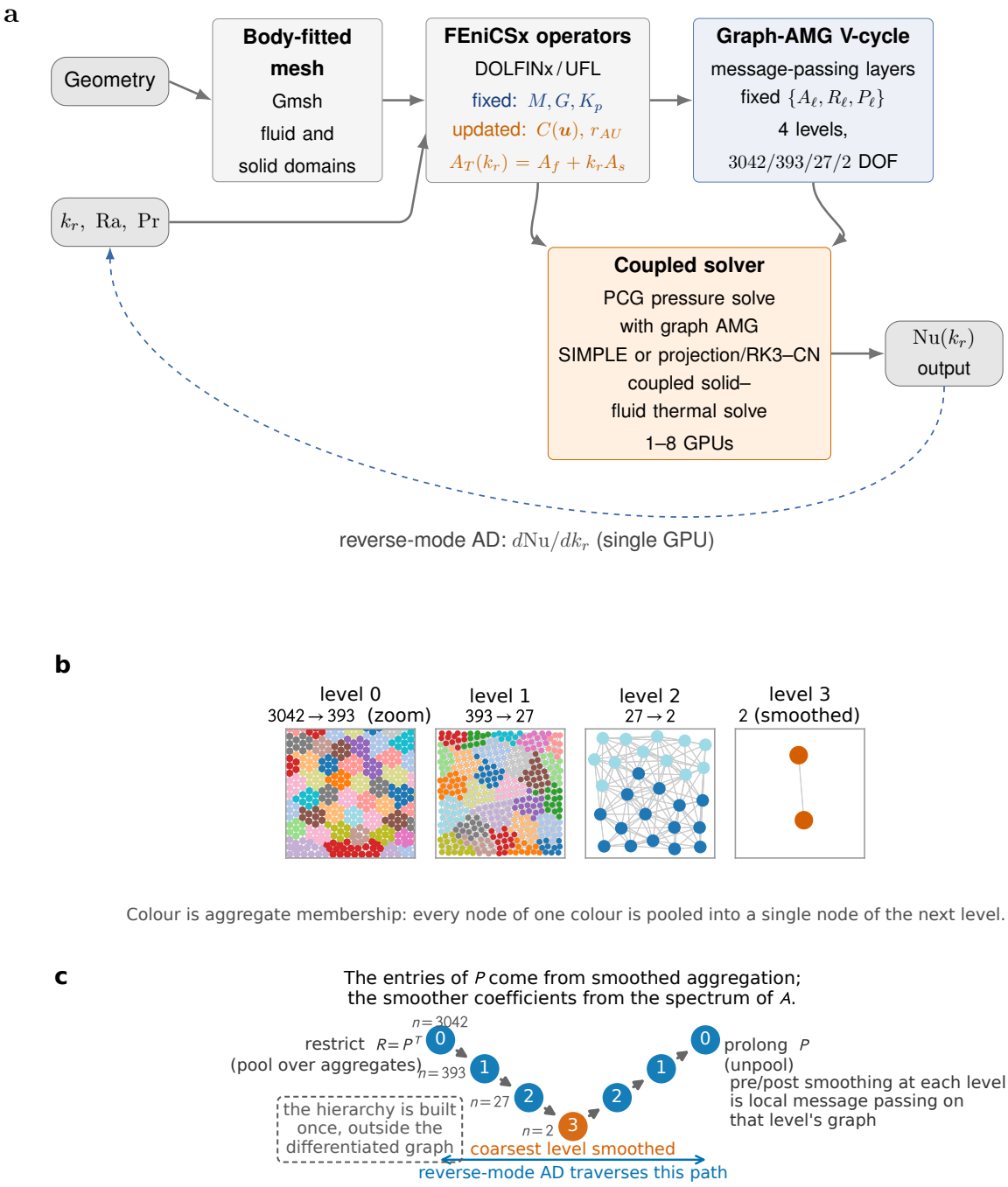


Figure 1. Algebraic multigrid executed as a fixed-weight graph U-Net. **(a)** The solver. Geometry-fixed operators and the multigrid hierarchy are blue; state- and parameter-dependent operators are orange, updated analytically and not trained. Reverse-mode differentiation traverses the executed pressure and thermal operations to give $d\mathrm{Nu}/dk_r$, on a single GPU. **(b)** The smoothed-aggregation hierarchy the code actually builds for the operator it preconditions, the P_1 pressure Laplacian on the body-fitted cavity mesh: the produced hierarchy, not a schematic. Level 0 is a zoomed quadrant because 3042 nodes are not separable by eye at full extent. **(c)** The same four levels as the encoder–decoder the GPU executes. Every weight is fixed by the discretization and the aggregation; none is trained. Methods gives the construction and the coarsening factors. Abbreviations used here and in later artwork: AD, automatic differentiation; MG, multigrid; AMG, algebraic multigrid; PCG, preconditioned conjugate gradient; UFL, Unified Form Language.

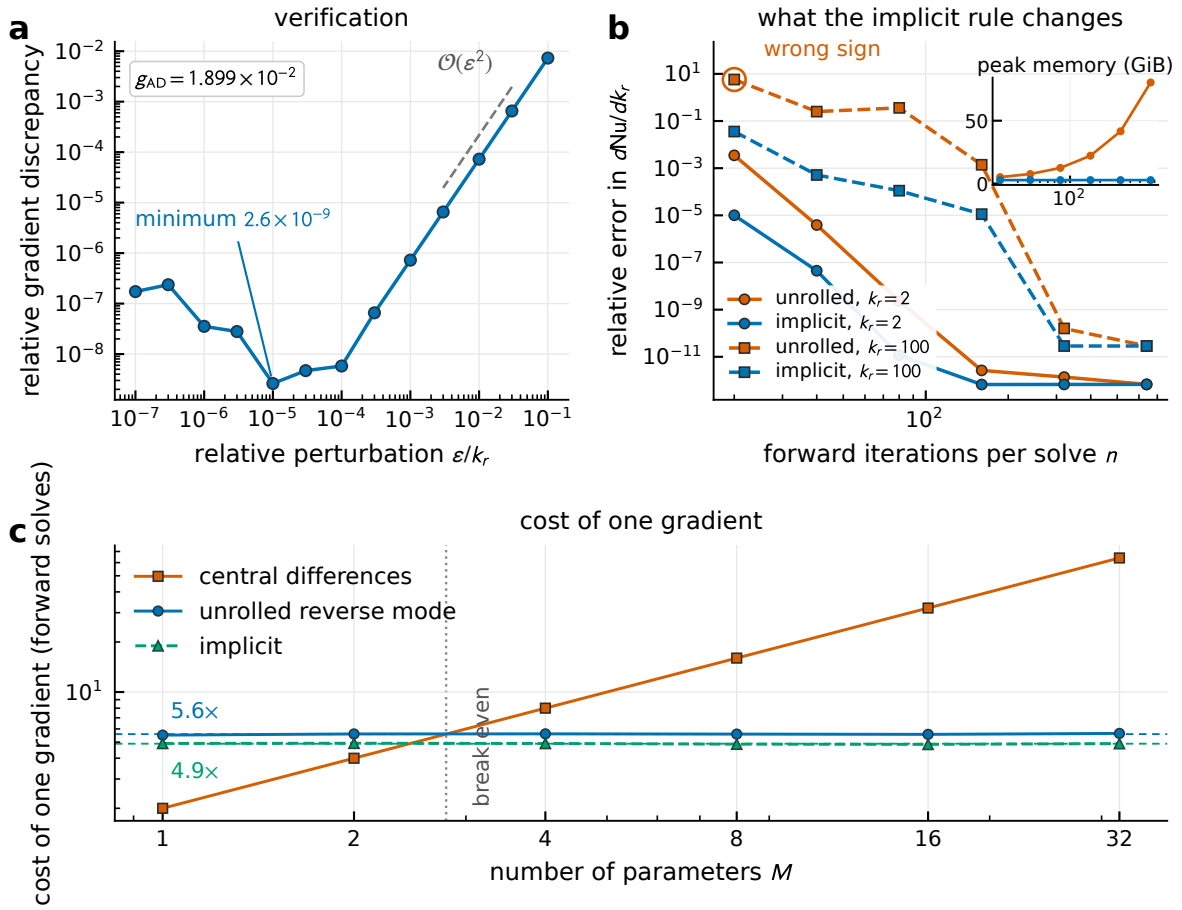


Figure 2. Gradient verification, and implicit differentiation against the truncated iteration. **(a)** Verification. For the conjugate cavity at $\mathrm{Ra} = 10^3$ and $k_r = 2$ the reverse-mode sensitivity is $g_{\mathrm{AD}} = 1.899 \times 10^{-2}$. The relative discrepancy against a central finite difference follows the expected $\mathcal{O}(\epsilon^2)$ truncation regime before round-off dominates, reaching a minimum of 2.6×10^{-9}. **(b)** What the implicit rule changes. Relative error in $d\mathrm{Nu}/dk_r$ against the forward iteration count n, at $k_r = 2$ and $k_r = 100$, measured against the converged-solve value both paths agree on at $n = 640$. Differentiating the truncated iteration is the upper curve of each pair at every count; at $k_r = 100$ and $n = 20$ it also has the wrong sign, and between $n = 40$ and $n = 80$ it gets worse before it gets better. The implicit rule is below it throughout and falls monotonically. At the prescribed forward count $n = 40$ of this differentiation experiment, the implicit gradient has a relative error of about 5×10^{-4} against 25% for the unrolled gradient at $k_r = 100$. *Inset:* peak memory over the same sweep, flat at 2.7 GiB for the implicit path against 5.1 to 80.4 GiB for the unrolled path, which stores every iterate. **(c)** Cost, in forward solves. Gradient times are normalized by the forward-solve time on the same GPU and mesh. Both reverse-mode routes are flat in M: unrolled at 5.6 and implicit at 4.9, moving 2.4% and 1.6% while M rises 32×, whereas a central difference costs $2M$. Unrolling breaks even near $M = 3$ and is 11.3× cheaper at $M = 32$; implicit reaches 13.1×. All three routes are the same solver on the same GPU, timed in one run. All panels are single-GPU, use fixed linear-solve iteration counts rather than the production stopping tolerance. The multigrid hierarchy is built once outside the differentiated graph; the unrolled route differentiates its application, while the implicit route uses a residual-based backward rule and a transposed solve without differentiating the iteration history (Supplementary Note S2).

only the residual and taking one transposed solve. At converged counts the implicit gradient reproduces the unrolled one to a relative 2×10^{-12}, both within 1.2×10^{-11} of a central finite difference. At the prescribed forward count of this experiment, where the solves are not converged, it is the more accurate: against the converged sensitivity it is 5×10^{-4} away at $k_r = 100$, where the unrolled gradient is 25% away. Its memory is independent of the iteration count, 2.7 GiB against the 41.6 GiB that the unrolled computational graph reaches at 320 iterations (Supplementary Note S2.4).

Using this gradient, Newton's method recovers the conductivity ratio in six iterations in the synthetic inverse problem, with k_r treated as unknown and one Nusselt number as data (Extended Data Fig. 2). This is a differentiability demonstration, not an optimization study. Both reverse routes cost a fixed number of forward solves whatever the parameter count, 5.6 unrolled and 4.9 implicit, against $2M$ for central differences (Figure 2c).

2.3 Wall-clock time to convergence

Supplementary Table S6 gives the wall-clock time to reach the common converged Nusselt number for every solver, mesh and Rayleigh number. The 2D wall-clock times are measured directly; the 3D times are reconstructed from iteration counts and cost per iteration. Conventions and tier labels are in Methods (Section 4.2). Meshes with the same label do not require the same computational effort, so the primary comparison is accuracy per cost (Figure 3).

Where the mesh resolves the flow, the four implementations agree. On the finest mesh the two P_2–P_1 solvers agree to three or four significant figures and the GNN matches cell-centred OpenFOAM v12 to within 0.15%. That residual is a discretization difference, not an artefact of OpenFOAM's stopping criterion, and grows as the mesh coarsens (Supplementary Note S8.4). The CNN overshoots Nu on coarse meshes, by up to +26% at 3D h_{16}, reflecting its lower-order boundary stencil (Supplementary Note S1), so wall-clock time is compared between solvers only on the mesh-converged cells.

Which solver converges fastest depends on regime and mesh size. OpenFOAM leads on the smallest meshes, where GPU kernels cannot amortize their launch overhead; the GNN leads on the larger 2D meshes and both convective 3D Rayleigh numbers; the FEniCSx reference is the bottleneck throughout (Extended Data Fig. 3a).

The memory behaviour shows the opposite trend (Extended Data Fig. 3b). At fixed accuracy the P_2–P_1 pair carries several times the peak GPU memory of the colocated CNN, 4.6× at $\mathrm{Ra} = 10^5$ and an order of magnitude at $\mathrm{Ra} = 10^6$, a trade of memory for time. All of this is Tier C (Section 4.2): the codes differ in more than the backend, so it is not the controlled ratio of Figure 3d,e.

2.4 Throughput on the turbulent benchmark: structured CNN, unstructured GNN, and CPU finite element method (FEM)

The turbulent Rayleigh–Bénard convection (RBC) cell challenges the solvers differently: a converged time-averaged $\langle \mathrm{Nu} \rangle$ requires marching many free-fall times through a chaotic flow (the $\Gamma = 2$, $\mathrm{Ra} = 10^7$ cell; the Johnston–Doering fit gives ≈ 13.6 [46]). We compare the explicit *CNN*, the unstructured *GNN* of this work, and a *CPU FEM* reference running the identical scheme on PETSc (Methods).

On wall-clock time to each solver's converged $\langle \mathrm{Nu} \rangle$ (Supplementary Note S3.2), a Tier D comparison sharing the problem and the quantity of interest but not the discretization (Section 4.2), the structured explicit CNN is faster: within seed scatter on the coarse mesh and 2.6× on the finer, because the chaotic flow forces a small advective step regardless. The two integrate to their *own* statistically stationary values, approaching a common $\approx$14.0 under refinement (Supplementary Note S3.2). We therefore do *not* claim a turbulent-RBC wall-clock win for the unstructured solver.

The per-step picture (Figure 3d,e) isolates one time step. The only controlled comparison is GNN vs. CPU FEM, which run the identical discretization at the same step and pressure tolerance, differing only in backend and hardware; the CNN step is context. The CNN step is cheapest ($\sim$7 ms); the FEM step is 12–23× costlier, and for that identical step the GPU is 2.2× faster than the 32-core CPU on the coarse mesh, rising to 3.6× on the finer. Carried to stationarity at h_{64} the CPU FEM run reaches $\langle \mathrm{Nu} \rangle = 14.62$ against the GNN's 14.64, so the two implementations are the same computation.

At $\mathrm{Ra} = 10^8$ (2D, h_{128}), our explicit CNN configuration diverges before the flow develops, whereas the semi-implicit GNN configuration (RK3–CN) remains stable on the same case (Supplementary Note S3). We claim only that our configuration fails where the semi-implicit solver is robust, not that explicit structured methods *generally* do. Single-GPU throughput at scale is in Extended Data Fig. 4, and its longer Supplementary note is in the code package (Code availability).

2.5 Conjugate heat transfer: validation and same-method throughput

An important class of natural-convection problems is *conjugate*: a heat-conducting solid immersed in the fluid, coupled to the flow only by conduction across an interface a body-fitted mesh represents without staircasing. The same solver extends to it without modification beyond the physics, solving the energy equation monolithically over the whole domain with a piecewise conductivity (Methods). The conjugate pathway is verified to $\mathcal{O}(h^3)$ in temperature (Supplementary Note S1.3). The transient benchmark uses the rotational incremental pressure-correction scheme (IPCS) with RK3–CN, and re-running

the same mesh sequence with an independent IMEX scheme, ARS(2,2,2) [47], reproduces the Nusselt number to at least five significant figures at every mesh (Table 1).

Against the transient conjugate cavity benchmark of Khatamifar et al. [48] the grid-converged GNN hot-wall Nusselt number lies inside the published inter-study spread in all six cases (Extended Data Table 2), with solution verification reported per Rayleigh number in Table 1. On the centred-conducting-body benchmark of House et al. [49], where the flow must route *around* a solid, the hot- and cold-wall Nusselt numbers agree to four significant figures. The grid-converged transient development is in Supplementary Note S4, and the flow structure and the partition-thickness sensitivity are in the code package (Code availability).

Throughput. The conjugate cavity affords the cleanest comparison here, because it is advanced with the same scheme and discretization however the linear systems are solved. We run it two ways differing in nothing but the linear solver and hardware: graph multigrid on one H200 GPU, and PETSc conjugate gradient (CG) with its geometric-algebraic multigrid (GAMG) preconditioner, to a true-residual 10^{-6}, on the CPU, holding mesh, time step, step count and the rotational-IPCS update identical. The GPU per-step cost is nearly mesh-independent (40–61 ms across a 31× size range) while the CPU sparse solve grows with size, so the advantage widens from 1.25× to 23.9× at 1.2×10^6 DOF (Figure 3d). This is Tier B (Section 4.2), matched in pressure *accuracy* but not in pressure *work*. The work imbalance runs *against* the GPU: the graph multigrid takes 10.3–16.3 CG iterations per solve where PETSc takes 8.8–11.5 (Supplementary Note S5.1), so the CPU baseline is not mis-preconditioned. Running the *identical* graph multigrid on the CPU is 4.6–21.4× slower than on the GPU (Tier A; Extended Data Table 3). At the finest mesh those two ratios, 21.4× and 23.9×, suggest device execution accounts for most of the difference; at the coarsest, 4.6× against 1.25×, it does not. Neither is a numerical factorization: the CPU graph path is not optimized. The same substitution gives 181× on the 3D conjugate heat sink at 1.2×10^7 coupled DOF, the largest matched pair in this work (Supplementary Table S14).

Wall-clock to a converged Nu**.** Both implementations integrate the identical discrete system, so the step count to a steady Nu ($|\Delta\mathrm{Nu}/\mathrm{Nu}| < 10^{-5}$) is shared, a property of the scheme not the linear solver. The GNN reaches steady state in under two hours at the finest mesh; the CPU reference is projected to need up to $\sim$43 hours (Supplementary Table S11).

2.6 An applied conjugate example: an engine cylinder

The same solver carries to an applied three-dimensional geometry: a single-cylinder internal-combustion engine in the configuration of a published industrial tutorial [50], where in-cylinder gas convects under buoyancy while the metal conducts. Where the reference workflow is partitioned, we solve one system and need no outer coupling loop. The nondimensional fire-deck temperature is grid-converged at $\theta = 0.1833$, corresponding to 527 K for a cast-iron head, a temperature that is physically plausible for an operating cast-iron cylinder head [51]: a plausibility check, not a validation against a reference solution.

This is the paper's only throughput comparison on curved body-fitted geometry, where the same substitution gives 23.9× at 7.4×10^5 coupled DOF against the CPU FEM reference, which shares the discretization exactly; the geometry and mesh study are in Supplementary Note S6 (Extended Data Fig. 5; Supplementary Table S13). The engine geometry figure and its distributed-scaling note are in the code package (Code availability).

2.7 Distributed capacity and the effect of coarse correction

The solver parallelizes by slab decomposition over the NVIDIA Collective Communications Library (NCCL), and the decomposition is trajectory-preserving: one-versus-two-GPU runs reproduce the single-GPU Nu to 8×10^{-11} on the unsteady Rayleigh–Bénard cell and 1.4×10^{-7} on the conjugate cavity (Table 1), both inside the pressure tolerance that bounds the comparison.

The distributed result is capacity: eight GPUs carry 1.18×10^8 DOF on the conjugate heat sink at a weak efficiency of 0.56 relative to one GPU (0.74 relative to two GPUs), and 4.7×10^8 on the engine geometry at 0.67 relative to one GPU (0.81 relative to two; Supplementary Table S22). The two-GPU reference excludes the one-to-two-GPU step, where the halo exchange and the coarse correction first appear. One mechanism makes that possible: a one-level block-Jacobi pressure preconditioner has no coarse level, so its iteration count grows with subdomain count, from 29.0 to 170.6 over one to eight GPUs on the fixed-size three-dimensional cavity (Supplementary Table S18). A two-level global coarse correction reduces that growth and turns a regressing strong-scaling curve into a positive one, 1.18 to 1.34× on eight GPUs across the three problems of Extended Data Fig. 1.

That is a statement about the mechanism, not efficient strong scaling: those speedups are near 15% parallel efficiency, against a deliberately simple block-Jacobi rather than a production distributed algebraic multigrid. The per-problem weak and strong tables are in Supplementary Note S7, and the coarse-space ablation and the engine mesh sequence are in the code package (Code availability).

3 Discussion

We separate same-discretization timing comparisons from comparisons of complete solver implementations. The ranking of the four solvers considered here depends on the regime, problem size, and the prescribed accuracy. For example, OpenFOAM v12 is fastest on the smallest meshes at loose tolerance, whereas the graph neural network solver is fastest on large problems at tight tolerance. We make no general claim that GPUs are superior to CPUs. Timings between different solvers are reported for orientation. They compare different implementations on different hardware and are not normalized for either (Section 4.2).

The numerical ingredients of the proposed approach are classical. Our contribution lies in the formulation, execution, and differentiability of the model. The resulting approach consists of a classical solver set within a differentiable framework. Finite element and multigrid operations are exactly represented, so the correspondence to the original discretized system is preserved. We also demonstrate the distinction between differentiating the executed algorithm and differentiating the discrete converged solution (Supplementary Note S2.3). Raw multigrid throughput is not the claim. On the pressure solve in isolation, NVIDIA AmgX is 1.39× faster than our graph multigrid at 1.68×10^7 DOF. Within a whole time step that advantage is diluted: on the assembled-Laplacian path AmgX is 0.5 to 4.7% faster per step, and on the Schur-complement cavities it is 3 to 42% slower (Supplementary Note S5.4).

The runtime covers the whole time step, including the pressure solve, projection and conjugate coupling, with no stand-alone linear-algebra library called during execution. Reverse-mode differentiation returns $d\mathrm{Nu}/dk_r$ directly and is consistent with a central finite difference computation of the gradient of the same fixed-iteration program (Section 2.2).

One gradient costs 5.6 forward solves with unrolling and 4.9 with implicit differentiation. Both costs are nearly constant from one to 32 parameters, whereas central differences cost two solves per parameter. For one or two parameters the finite difference is cheaper. The two routes break even near three parameters, and at 32 the reverse-mode gradient is 11.3× cheaper unrolled and 13.1× cheaper implicit. Against pyadjoint on DOLFINx the unrolled reverse mode is more expensive per forward solve, because it differentiates through the linear-solver iterations, whereas pyadjoint solves an adjoint linear system. The faster forward solve offsets this overhead: the delivered gradient is 2.2 to 5.9× faster (Supplementary Table S4, the unrolled comparison).

The clearest controlled result is the comparison of two solvers using the same discretization, with only the backend and execution device changed (Tier B, Section 4.2). Across the four matched problems of Table 2 and Figure 3d, the ratio of CPU to GPU time per step spans nearly three orders of magnitude, from below one on the smallest

engine mesh, where the GPU is slower, to 181× on the largest mesh. For this fixed hardware pair, the CPU/GPU time ratio increases with problem size. Running the identical fixed-weight solver on both backends gives 21.4× against the 23.9× same-method ratio (Supplementary Note S5.2).

The 181× ratio measures the complete time step; the pressure solve alone is 18.7× faster. The pressure solve is a small part of the step in both implementations, and on that phase alone PETSc GAMG takes *fewer* iterations than ours. Most of the ratio comes from the implicit temperature, velocity and corrector solves, which run the same CG + Jacobi in both implementations (Table 3). The large number measures a time step whose state remains GPU-resident, not a better multigrid.

The value of the unstructured solver is not the raw speed where a Cartesian grid suffices, but the demonstration that a formulation based on a neural network can run on body-fitted geometry whilst retaining finite-element accuracy (Section 2.6). On the turbulent RBC cell, which suits a Cartesian grid, the structured CNN is faster, with many inexpensive explicit steps outperforming a smaller number of more expensive semi-implicit steps (Supplementary Note S3.2).

The distributed study is scoped to one node of eight H200 GPUs. For the steady three-dimensional cavity, a one-degree-per-rank Nicolaides deflation [34] achieves the best iteration count, wall time and speed-up at two, four and eight GPUs, without replicating the coarse operator (Supplementary Table S18). This study is intended to assess the scaling behaviour of the proposed approach, rather than to provide a state-of-the-art distributed pressure solver.

There are limitations to this study. The two-level coarse correction is replicated on every rank, which is viable on one node but reintroduces at multi-node scale the memory cost that deflation avoids. The CPU baselines are not tuned to their optima, and we ran no structured solver on a curved-interface case, so the cost of staircasing is unquantified. Unrolled differentiation through the linear solver stores every iterate on the computational graph, so its memory grows by about 2.6× each time the mesh is doubled. The unrolled conjugate thermal study therefore stops at 9.4×10^4 DOF on a single 141 GB GPU. Implicit differentiation records only the converged residual, so its memory does not depend on the iteration count (Supplementary Note S2.4). In this comparison, pyadjoint avoids the iteration-history storage that unrolling requires. The implicit implementation also avoids that storage, but its runtime is not compared with pyadjoint in Supplementary Table S4. Two structural limitations remain: the demonstration is single-GPU because the in-place halo exchange breaks the computational graph, and the optional Picard momentum path is a nonlinear fixed point whose adjoint is that of the nonlinear system. Finally, in one benchmark case the CPU FEM reference sits 0.35% below the value shared by the other three solvers and the published benchmark. The two

P_2–P_1 codes assemble bit-identical operators, so incomplete convergence of that run is the suspected cause. We report it as an unresolved outlier.

The benchmark is intended to establish principles that extend beyond these particular results. In the future, as accelerators and learned components enter simulation, meaningful speedup comparisons require the numerical method to be held fixed while execution is varied in isolation. Tiers A and B provide this controlled comparison.

4 Methodology

4.1 Governing equations and numerical overview

Domains and unknown fields. Let $d \in \{2, 3\}$ be the spatial dimension. The fluid occupies $\Omega_f \subset \mathbb{R}^d$. For conjugate heat transfer, the solid occupies $\Omega_s \subset \mathbb{R}^d$. The two domains are disjoint and meet at the interface

$$\Gamma_{fs} = \partial\Omega_f \cap \partial\Omega_s.$$

The fluid unknowns are the velocity $\mathbf{u} : \Omega_f \times (0, t_f] \to \mathbb{R}^d$, the pressure $p : \Omega_f \times (0, t_f] \to \mathbb{R}$, and the temperature $T_f : \Omega_f \times (0, t_f] \to \mathbb{R}$. The solid temperature is $T_s : \Omega_s \times (0, t_f] \to \mathbb{R}$. Thus, $\mathbf{u}$ is a vector field. The pressure and temperatures are scalar fields.

Boussinesq model. The fluid is assumed to be incompressible and Newtonian. Its properties are constant, except for the density in the buoyancy force. Under the Boussinesq approximation,

$$\rho(T^*) = \rho_0 \left[1 - \beta \left(T^* - T_0\right)\right], \qquad \beta\Delta T \ll 1,$$

where T^* is the dimensional temperature. Density variations are neglected in the continuity, inertia, and diffusion terms. They are retained only in the buoyancy term. The hydrostatic pressure is absorbed into the modified pressure p. Viscous dissipation is neglected. Volumetric heat sources are neglected except in the conjugate heat-sink case, where a uniform nondimensional heating $q = 10^4$ is prescribed in a small solid source block at the base of the heat sink and enters Equation (2) as a source term $q\,\chi_{\mathrm{src}}$ on the right-hand side, with χ_{src} the indicator of the source block (Supplementary Note S7.2). The solid is rigid and stationary.

Lengths are scaled by L, velocity by κ_f/L, time by L^2/κ_f, pressure by $\rho_0\kappa_f^2/L^2$, and temperature by ΔT. Here, κ_f is the thermal diffusivity of the fluid. Scaling time on the fluid diffusive scale L^2/κ_f is what makes the diffusion coefficient in Equation (1c) unity and leaves Pr on the viscous term of Equation (1b); scaling on the viscous time L^2/ν

would exchange the two. After dropping dimensionless superscripts, the fluid equations are

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{in } \Omega_f \times (0, t_f], \tag{1a}$$

$$\partial_t \mathbf{u} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot (2\mathrm{Pr}\, \boldsymbol{\varepsilon}(\mathbf{u})) + \mathrm{Ra}\,\mathrm{Pr}\, T_f\, \widehat{\mathbf{e}}_b \qquad \text{in } \Omega_f \times (0, t_f], \tag{1b}$$

$$\partial_t T_f + \mathbf{u} \cdot \nabla T_f = \nabla^2 T_f \qquad \text{in } \Omega_f \times (0, t_f]. \tag{1c}$$

Here,

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathsf{T}}\right)$$

is the symmetric strain-rate tensor. Both $\mathbf{u} \otimes \mathbf{u}$ and $\boldsymbol{\varepsilon}(\mathbf{u})$ are second-order tensors whose divergences are vectors. The vector $\widehat{\mathbf{e}}_b$ points in the opposite direction to gravity. The dimensionless parameters are

$$\mathrm{Pr} = \frac{\nu}{\kappa_f}, \qquad \mathrm{Ra} = \frac{g\beta\Delta T L^3}{\nu \kappa_f}.$$

We use $\mathrm{Pr} = 0.71$ for air. For constant viscosity, incompressibility (Equation (1a)) collapses the symmetric-gradient viscous term,

$$\nabla \cdot (2\mathrm{Pr}\, \boldsymbol{\varepsilon}(\mathbf{u})) = \mathrm{Pr} \nabla^2 \mathbf{u}.$$

The pressure is defined up to an additive constant, which is fixed by pinning one P_1 degree of freedom, with the right-hand side mean-subtracted for compatibility.

For conjugate heat transfer, the solid temperature satisfies

$$c_r\, \partial_t T_s = \nabla \cdot (\mathbf{K}_r \nabla T_s) \qquad \text{in } \Omega_s \times (0, t_f], \tag{2}$$

where

$$c_r = \frac{(\rho c_p)_s}{(\rho c_p)_f}$$

is the scalar volumetric heat-capacity ratio. Equation (2) is scaled with the same reference quantities as the fluid equations, so the fluid conductivity, density and specific heat enter it only through the denominators of c_r and k_r. Using one set of scales across both subdomains is what leaves the interface conditions free of additional factors. The quantity $\mathbf{K}_r$ is the dimensionless solid conductivity tensor. It is a symmetric positive-definite second-order tensor. The present study uses isotropic materials, for which

$$\mathbf{K}_r = k_r \mathbf{I}, \qquad k_r = \frac{k_s}{k_f},$$

where k_r is a scalar conductivity ratio and $\mathbf{I}$ is the identity tensor. Here k denotes thermal conductivity and κ thermal diffusivity, related by $\kappa = k/(\rho c_p)$; they are distinct quantities, and κ_f is the fluid diffusivity that sets the scaling above and enters Pr and Ra.

At the stationary solid–fluid interface, we impose

$$\mathbf{u} = \mathbf{0}, \qquad \text{on } \Gamma_{fs}, \tag{3}$$

$$T_f = T_s, \qquad \text{on } \Gamma_{fs}, \tag{4}$$

$$\mathbf{q}_f \cdot \mathbf{n}_f + \mathbf{q}_s \cdot \mathbf{n}_s = 0, \qquad \text{on } \Gamma_{fs}, \tag{5}$$

where

$$\mathbf{q}_f = -\nabla T_f, \qquad \mathbf{q}_s = -\mathbf{K}_r \nabla T_s$$

are heat-flux vectors. The vectors $\mathbf{n}_f$ and $\mathbf{n}_s$ are the outward unit normals of the fluid and solid domains.

The zero jump above holds where the interface carries no source. The engine case of Supplementary Note S6 does not: a prescribed cycle-averaged wall flux q'' is imposed on the bore surface, which is itself Γ_{fs}, so the condition there is

$$\mathbf{q}_f \cdot \mathbf{n}_f + \mathbf{q}_s \cdot \mathbf{n}_s = -q'' \qquad \text{on } \Gamma_{fs}, \tag{6}$$

with $q'' > 0$ a heat input. The sign follows from the weak form. Integrating the conduction term by parts on each subdomain and summing leaves the interface contribution $-\int_{\Gamma_{fs}} (\mathbf{q}_f \cdot \mathbf{n}_f + \mathbf{q}_s \cdot \mathbf{n}_s)\, v \, \mathrm{d}S$ on the right-hand side, so a source assembled as $+\int_{\Gamma_{fs}} q''\, v \, \mathrm{d}S$ requires the jump $-q''$. The same reading follows from the normals: $\mathbf{n}_f$ and $\mathbf{n}_s$ both point out of their subdomains, so a positive jump is heat leaving both sides, and injecting heat therefore makes the jump negative. The jump is the imposed source rather than zero. Because the discrete temperature is continuous across Γ_{fs}, one P_2 field spanning both subdomains, the source is assembled once on the shared interface degrees of freedom, as $\int_{\Gamma_{fs}} q'' \{v\} \, \mathrm{d}S$ with $\{\cdot\}$ the facet average, and is not attributed to either side. It is then carried away by conduction into the metal and by conduction and convection into the gas, in proportion to the local conductivities. The reference workflow instead passes a gas-side wall flux across a partitioned interface, so depositing it at the interface is a modelling choice of the monolithic formulation, not a restatement of the partitioned one.

Benchmark configurations. We consider five configurations. The differentially heated cavity has isothermal hot and cold side walls, adiabatic remaining walls, and no-slip velocity conditions [44, 52]. The Rayleigh–Bénard cell has aspect ratio $\Gamma = 2$, periodic horizontal boundaries, and isothermal no-slip plates. The conjugate cavity contains both fluid and solid domains, separated by a central conducting partition, and satisfies the interface conditions above. Further details are given in Section 2.5. Two further conjugate cases are used for the distributed study. A branched conducting heat sink ($k_s/k_f = 100$) sits in a buoyant air-filled enclosure ($\mathrm{Ra} = 10^6$), in the spirit of topology-optimized natural-convection designs [53]; its thin branches must be resolved,

so a converged source-averaged temperature ($\theta_{\mathrm{src}} \approx 2.36$) needs $\gtrsim 7 \times 10^6$ DOF (Supplementary Table S21 and Supplementary Fig. S5). The engine cylinder of Supplementary Note S6 is the fifth.

Turbulent comparison. On the turbulent Rayleigh–Bénard cell three solvers span the structured/unstructured and CPU/GPU axes. The explicit CNN takes $\sim 10^6$ tiny steps. The GNN takes $\sim 10^5$ larger implicit steps on the GPU. The CPU FEM reference runs the *identical* RK3–CN scheme on the *identical* discretization but solves with PETSc (CG + GAMG) on 32 cores, making it the transient analogue of the conjugate same-method comparison.

The Nusselt number is computed from the heat flux through the active boundary Γ_a:

$$\mathrm{Nu}(t) = \frac{\left| \int_{\Gamma_a} \nabla T(t) \cdot \mathbf{n} \, \mathrm{d}\Gamma \right|}{\int_{\Gamma_a} \mathrm{d}\Gamma}, \tag{7}$$

the wall-averaged nondimensional temperature flux through the active wall Γ_a, evaluated as a consistent finite element functional of the discrete temperature. The cavity and conjugate walls have unit length in this nondimensionalization, so there the denominator is 1; the $\Gamma = 2$ Rayleigh–Bénard plates do not, and the average is what is reported. With unit temperature difference the pure-conduction value is 1 for the fluid-only cavity. For the conjugate cases we follow the convention of Khatamifar et al. [48], whose tabulated values this matches; there the composite conduction flux is not unity, so $\mathrm{Nu} = 1$ does not mark pure conduction. The Nusselt number is a scalar post-processed quantity. For steady cases, we report its converged value. For transient cases, we report its time history or its average over the specified sampling interval.

Conjugate formulation. The energy equation is solved monolithically over the whole domain with a piecewise conductivity, $k_s/k_f = k_r$ in the solid band, where $\boldsymbol{u} \equiv \mathbf{0}$. The two phases carry equal volumetric heat capacity, per the benchmark convention; a distinct solid heat capacity is left to future heterogeneous-material studies. Temperature continuity holds by construction of the C^0 space, while flux continuity is enforced weakly through the Galerkin natural boundary term. The discrete normal-flux jump is therefore not identically zero, but it converges at the finite element rate.

Numerical overview. The steady cavity calculations use a segregated SIMPLE pressure-correction method [1]. The transient finite element calculations use a rotational IPCS [54] with semi-implicit third-order Runge–Kutta/Crank–Nicolson time integration, denoted RK3–CN. The transient CNN uses an explicit predictor–corrector method with

pressure projection. The complete update equations, time-step restrictions, and linear-solver settings are given in the Supplementary Information. All computations use double precision.

Shared controls and stopping test. The four solvers are tied together by a common set of high-level controls. The SIMPLE runs share an under-relaxation schedule: α_U up to 0.7, reduced to 0.5 at $\mathrm{Ra} \geqslant 10^5$ and to 0.3 at $\mathrm{Ra} \geqslant 10^6$, with $\alpha_p = 0.3$ and $\alpha_T = 0.9$. One exception is disclosed. The finest three-dimensional FEniCSx cell at $\mathrm{Ra} = 10^4$ uses $\alpha_U = 0.5$, because 0.7 is marginal there (Supplementary Tables S6 and S23). The steady cases stop on a common test: $|\Delta\mathrm{Nu}/\mathrm{Nu}| < 10^{-5}$ sustained over three consecutive mesh-scaled iteration windows, each of order $2N$ SIMPLE steps and applied after a mesh-dependent warm-up. The conjugate cases apply the same 10^{-5} tolerance on physical-time windows rather than iteration counts. The turbulent Rayleigh–Bénard case cannot use either, because Nu does not settle to a value: there we require the *window-mean* Nu to drift by less than 5×10^{-2} over three consecutive windows of 0.05 thermal-diffusion times (≈ 133 convective turnovers each), after an equal warm-up and a minimum averaging span of 0.25 diffusion times. That tolerance is roughly three times the measured window-to-window scatter of the mean, so it is a drift test rather than a convergence test; all reported Rayleigh–Bénard wall-clock times are set by it. For OpenFOAM the test is applied to its wall-heat-flux trace by an external monitor that terminates the run. Its inner linear solves keep their native tolerances, and the reported results are insensitive to those tolerances (Supplementary Table S25). The CNN, GNN, and FEniCSx use the same plain SIMPLE iteration body. No outer-loop acceleration is applied to any of them, apart from two disclosed three-dimensional FEniCSx cells: the $\mathrm{Ra} = 10^4$ cell noted above ($\alpha_U = 0.5$) and the $\mathrm{Ra} = 10^3$ cell, which uses an Anderson-accelerated iteration count (Supplementary Table S6). Harmonizing these controls makes the cross-solver cavity timings a complete-solver-stack comparison at matched high-level settings, which is Tier C.

Proposed GPU-native GNN solver. The main method proposed in this work is the GPU-native GNN solver. FEniCSx provides the finite element definition and assembly backend through DOLFINx and UFL [3]. The graph execution is therefore not intrinsically tied to one finite element space. To keep the present study focused, we use the mixed P_2–P_1 Taylor–Hood pair for velocity and pressure [26]. The domains are discretized with body-fitted Gmsh meshes [55].

The assembled finite element operators are transferred to the GPU runtime. Each linear operator is applied as a sparse matrix–vector product, which is weighted message passing on the degree-of-freedom graph:

$$(\mathsf{A}x)_i \;=\; \sum_{j\in\mathcal{N}(i)} \mathsf{A}_{ij}\, x_j, \tag{8}$$

where $\mathcal{N}(i)$ is the set of degrees of freedom coupled to i by a nonzero of A. Nodes are degrees of freedom, edges are nonzero couplings, and edge weights are entries of the assembled finite element operators. The identity is exact, not an approximation, which is what lets the discrete method carry over unchanged.

The pressure multigrid hierarchy is constructed algebraically from the assembled pressure Laplacian. Its smoothing, restriction, prolongation, and coarse-grid operations form a graph U-Net (Figure 1b,c). Restriction pools over the aggregates, prolongation unpools, and smoothing is local message passing on each level's graph. For the instance drawn in Figure 1, the P_1 pressure Laplacian on the body-fitted cavity mesh, the hierarchy has four levels carrying 3042, 393, 27 and 2 unknowns, coarsening by factors 7.7, 14.6 and 13.5 at an operator complexity of 1.25. The coarsest level is relaxed by the same smoother rather than solved directly, so the whole cycle is one kind of operation applied on a sequence of graphs. The multigrid V-cycle preconditions an outer conjugate-gradient solve. All weights are fixed by the finite element discretization or algebraic interpolation. Nothing is trained as any weights of CNN and GNN are set in advance using coefficients derived from the numerical discretizations and methods chosen. State- and parameter-dependent operators are updated analytically from the current solution. The momentum, pressure, and thermal operations remain within the same GPU-native computational graph.

Reference implementations. Three solvers provide comparisons with the proposed GNN. The CNN follows the fixed-weight Neural Physics construction [11, 13] on a uniform Cartesian grid. The CPU FEniCSx solver uses the same finite element forms and Taylor–Hood discretization as the GNN, with PETSc linear algebra [3, 56]. This is the closest match to the methods used in the GNN solver, and is the same-method reference throughout. OpenFOAM v12 provides a production cell-centred finite volume reference on the CPU [2, 43]. Detailed implementation and benchmark settings are given in the Supplementary Information.

Verification. The proposed GNN solver is verified with the method of manufactured solutions before the performance tests. It attains the expected Taylor–Hood spatial orders in both single-GPU and distributed multi-GPU configurations. The CNN solver is tested with the same manufactured solutions. Refining the grid reduces the error until the boundary stencil prevents further improvement. The full convergence studies and implementation details are reported in Supplementary Note S1.

4.2 Measurement conventions and comparison tiers

All four solvers advance the same dimensionless Boussinesq system under one shared high-level algorithm: a common SIMPLE under-relaxation schedule, a common converged-Nusselt target on the steady cavity problem, and the same 10^{-6} pressure tolerance wherever a linear solve is compared directly. One finest-mesh 3D FEniCSx cell departs from that schedule, as noted above. What differs is the discretization, the linear solver and the hardware. In a matched comparison the two implementations differ only in the keys stated for that comparison, and a run ends when the named quantity meets its stated stopping tolerance. A *mesh label* h_n names the mesh with n nominal cells per unit edge, hence n^2 cells in two dimensions and n^3 in three; the same label is used for both, and larger n is finer. The realized cell size is a different quantity, written $\bar{h}$ and tabulated where the distinction matters, because on the unstructured triangulations it departs from $1/n$. The conjugate and heat-sink studies refine mesh sequences that have no cells-per-edge value at all; those keep the names m_1 to m_6, and coarse, medium and fine for the engine, and their tables give the degree-of-freedom count against every entry rather than a label that would imply a uniform resolution they do not have. Three conventions then hold throughout. First, cross-solver timings are reported for *orientation*, not as hardware-normalized comparisons. The CPU references (FEniCSx and OpenFOAM on 32 cores; PETSc on 32 cores) are stated configurations, not each code's tuned optimum. Where we did measure thread scaling, on the body-fitted engine and the conjugate thermal step, doubling the allocation from 16 to 32 cores changed the CPU time by -3 to $+1$% on the engine and by -3 to $+6$% on the conjugate thermal step (Supplementary Tables S13 and S4), so PETSc CG + GAMG does not thread-scale on those problems; we did not measure it for the cavity and Rayleigh–Bénard references. The reported per-step ratios, from 0.81 on the coarsest engine mesh to 181 on the largest heat-sink mesh, are therefore specific to this 32-core baseline and are not intrinsic constants. Second, timings are labelled *measured*, *reconstructed* (iterations $\times$ isolated cost per step) or *projected* (a measured GPU run scaled by the per-step ratio), and the three are never combined inside one ratio. Third, cross-solver *accuracy* is read from discretization-invariant functionals: the wall Nusselt number for temperature, the centreline extrema for velocity, and the conserved face-normal ($H(\mathrm{div})$) flux where a velocity field is compared across discretizations.

Comparison tiers. The comparisons in this study hold different amounts fixed, and only the tightest support causal statements about execution. We therefore label each comparison A–D by what it holds fixed and what it supports, and cite the tier at each use. Tier A holds the code and operators fixed and therefore isolates device execution. Tier B holds the discretization and time step and isolates the linear-algebra backend and the device, so it bounds a combined effect. Tier C holds the PDE and the outer controls

and measures the complete solver stack. Tier D holds only the physical problem and the quantity of interest and measures practical time to result. Tiers C and D are therefore end-to-end or practical measures rather than causal attributions.

4.3 Structured-grid CNN and the proposed unstructured-mesh GNN

Both GPU solvers target the same Boussinesq model introduced in Section 4.1, but they realize it in different numerical forms (Extended Data Fig. 6). The structured-grid CNN solver uses finite differences on a uniform Cartesian grid. The proposed GNN uses finite elements on a body-fitted unstructured mesh. Both are fixed-operator solvers where the weights of the neural networks are determined by the numerical methods used so no training is performed.

The CNN follows the Neural Physics construction [11, 13]. Analytical 3×3 convolution kernels apply the finite difference operators on a Cartesian grid. Boundary conditions are applied by ghost-layer padding. Advection uses a nine-point stencil. For the steady cavity, the CNN uses SIMPLE with a convolutional multigrid-preconditioned conjugate-gradient pressure solve. For the transient Rayleigh–Bénard case, it uses an explicit two-stage Heun predictor–corrector method with a standalone convolutional-multigrid pressure projection. That projection is an F-cycle with a 3×3 Jacobi-convolution smoother, stride-2 restriction and bilinear prolongation. This gives a low cost per step, but the stable step size is restricted by the explicit treatment.

The GNN construction, its graph-AMG pressure preconditioner, and the reference implementations are described in Section 4.1. The graph weights are fixed by the finite element discretization or by the algebraic interpolation. The equivalence is algebraic rather than approximate: the message-passing layers apply the identical iteration matrix, so the graph U-Net reproduces the iterates of the algebraic-multigrid V-cycle it re-expresses. State- and parameter-dependent operators, including the advection operator $\mathsf{C}(\mathbf{u})$ and the thermal operator $\mathsf{A}_T(k_r)$, are updated analytically from the current solution and the current parameters.

For steady cases, the proposed GNN uses the SIMPLE procedure of Supplementary Equations (S5) and (S6). The transient scheme is defined next.

The transient GNN step. Time advance is a low-storage three-stage Runge–Kutta scheme for advection and buoyancy with Crank–Nicolson (CN) diffusion, using the substep weights $\alpha = (\frac{8}{15}, \frac{5}{12}, \frac{3}{4})$, $\beta = (0, -\frac{17}{60}, -\frac{5}{12})$, $\gamma = (\frac{8}{15}, \frac{2}{15}, \frac{1}{3})$ with $\sum_i \gamma_i = 1$ [57]. The mass M, stiffness K, and gradient G matrices are assembled once and thereafter applied as sparse matrix–vector products on the GPU [58]. Each of the three substeps i executes:

1. **Nonlinear advection and buoyancy.** The advection operator is re-assembled on the GPU at the current substep state and applied to (u, v, T). Buoyancy enters v as the body force $\mathrm{Ra}\,\mathrm{Pr}\,\mathsf{M}\,T$. A two-register low-storage combination $\alpha_i F^{(i)} + \beta_i F^{(i-1)}$ forms the explicit part.

2. **Crank–Nicolson diffusion solve.** The viscous and thermal terms are implicit. Each substep solves the symmetric positive-definite systems

$$\begin{aligned}\big(\mathsf{M} + \tfrac{1}{2}\gamma_i \Delta t\,\mathrm{Pr}\,\mathsf{K}\big)\mathbf{u}^* &= \mathsf{M}\mathbf{u}^{(i-1)} + \Delta t\big(\alpha_i F^{(i)} + \beta_i F^{(i-1)}\big) \\ &\quad - \gamma_i \Delta t\,\mathsf{G} p^{(i-1)} - \tfrac{1}{2}\gamma_i \Delta t\,\mathrm{Pr}\,\mathsf{K}\mathbf{u}^{(i-1)}, \\ \big(\mathsf{M} + \tfrac{1}{2}\gamma_i \Delta t\,\mathsf{K}_T\big)T^* &= \mathsf{M}T^{(i-1)} + \Delta t\big(\alpha_i F_T^{(i)} + \beta_i F_T^{(i-1)}\big) - \tfrac{1}{2}\gamma_i \Delta t\,\mathsf{K}_T T^{(i-1)},\end{aligned}$$

   by conjugate gradients, with K_T the conductivity-weighted energy operator. Both systems are evaluated against the previous *substep* state $(\cdot)^{(i-1)}$, which equals the start-of-step state only at $i = 1$; the energy system carries no pressure term. The lagged pressure gradient and the explicit Crank–Nicolson half are both on the right-hand side. Treating the diffusion implicitly removes the diffusive step-size limit.

3. **Rotational incremental-pressure projection.** With $\mathsf{G}_{d,ij} = \int \phi_i\, \partial_d \psi_j$, the quantity $\mathsf{G}^\top \mathbf{u}^*$ is the negative of the weak divergence, and the signs below follow that convention. It sources the pressure-Poisson problem $\mathsf{K}_p\, \delta p = \mathsf{G}^\top \mathbf{u}^* / (\gamma_i \Delta t)$, solved by the graph multigrid. The velocity is corrected by a consistent mass-matrix projection $\mathbf{u} \leftarrow \mathbf{u}^* - \gamma_i \Delta t\, \mathsf{M}^{-1}\mathsf{G}\,\delta p$, and the pressure is updated with the rotational term $p \leftarrow p + \delta p + \mathrm{Pr}\,\mathsf{M}_p^{-1}\mathsf{G}^\top \mathbf{u}^*$, with M_p the lumped P_1 mass matrix.

The advective Courant–Friedrichs–Lewy (CFL) limit is comparable to the CNN's. The implicit CN treatment of diffusion and the larger semi-implicit substep let the solver reach the same statistics in far fewer, individually more expensive steps. In this way the same body-fitted spatial discretization and graph-AMG pressure solver carry from steady natural convection to transient Rayleigh–Bénard flow and conjugate heat transfer.

The proposed GNN also extends to distributed execution. The mesh is partitioned across GPUs, halo values are exchanged by NCCL, and the pressure solve uses distributed coarse correction. This allows the same solver structure in Extended Data Fig. 6 to scale from one to eight GPUs.

The reference implementations are described in Section 4.1. Their run settings, the solver configuration, and the hardware and software stack are given in Supplementary Note S8.

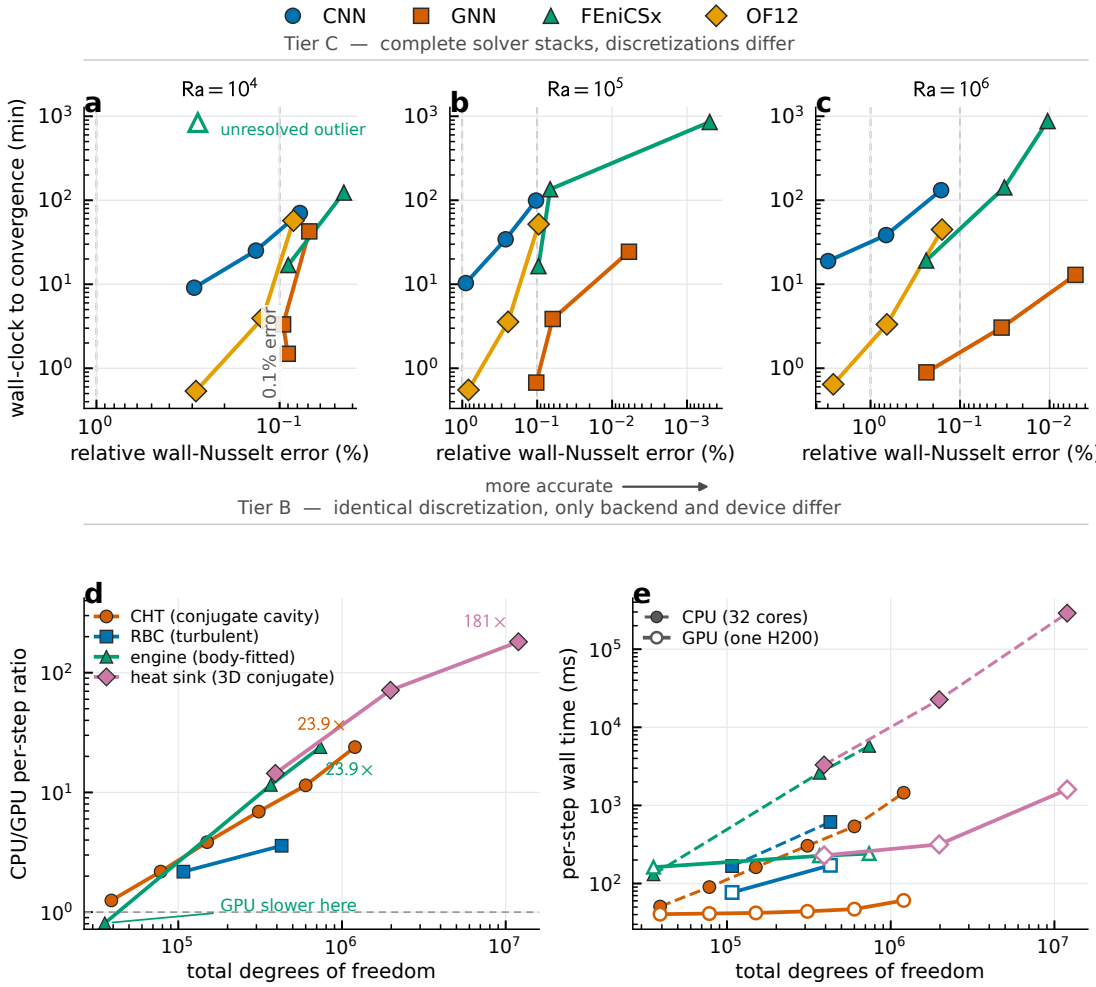


Figure 3. Work–precision (Tier C) and same-discretization throughput (Tier B). The rows answer different questions and are not to be read as one result; each is labelled with the tier it belongs to (Section 4.2). **(a–c) Accuracy per cost on the 2D cavity, Tier C.** Wall time to the common convergence criterion against the converged relative wall-Nusselt error e_{Nu}, on unrounded Nusselt values, at the $h_{64}/h_{128}/h_{256}$ meshes (references as in Table 1). Axes are logarithmic and smaller error lies to the right. Interquartile ranges are drawn where three or more repeats exist and are smaller than the marker throughout; the CPU references and seven of the eighteen GPU cells are single runs. The open green triangle at h_{256}, Ra $= 10^4$, labelled unresolved outlier, is excluded from the trend; incomplete convergence of that run is the suspected cause, not a demonstrated one (Extended Data Table 1). Being Tier C, the discretizations differ, so these panels measure complete solver stacks rather than isolating any one axis. Which solver reaches a given accuracy target first, and peak GPU memory over the same sweep, are in Extended Data Fig. 3a and Fig. 3b. **(d–e) Same-discretization CPU-versus-GPU throughput, Tier B, is a large-problem effect.** Per-step wall time of the *identical* P_2–P_1 rotational-IPCS RK3–CN scheme, graph multigrid on one GPU against PETSc on 32 CPU cores, for the four problems of Table 2, which gives the values and the per-problem sources. Ratios are not combined across problems. In **(d)** the ratio rises with problem size in all four, from 0.81 on the smallest engine mesh, where the GPU is the slower of the two, to 181× on the largest heat-sink mesh; the heat-sink and engine mesh sequences overlap in size and agree there, so size, not the hardware pair, is the controlling variable. **(e)** shows why: the GPU cost grows much more slowly with problem size than the CPU cost. The per-step ratio equals the wall-clock ratio at fixed step count.

Data availability

The benchmark meshes, the resolved per-run configuration files, the per-run result files (JSON, each carrying its resolved configuration and a configuration hash), and the processed Nusselt-number, wall-clock and scaling tables underlying every figure and table in this work will be deposited in a public repository with a citable DOI on publication. These artefacts are independent of the solver source and are released in full. Supplementary Note S8.6 lists the deposit contents item by item.

Code availability

The production solver implementations are proprietary and are not released. The code package `gnn4buoyancy-mwe` is public under the MIT licence at https://github.com/teeratornk/multigrid-gnn. This submission uses its release tag `v1.2.2`, commit `c9fd904b3cb3`, which the editor and reviewers can run and inspect. It contains a minimal working example of the fixed-weight graph-multigrid pressure solve expressed as message-passing layers, and runnable experiments on the package's own example operators that check the graph execution against the multigrid algebra it represents. It also holds the tables, figures and text moved out of the Supplementary Information. The package installs from a locked environment with uv, takes every parameter from Hydra configuration files, and carries a test suite. It does not reproduce the full benchmark. An archived copy of the release will receive a citable DOI on publication. The discrete equations, the settings of each of the four solvers and the hardware and software stack are given in Methods and in Supplementary Note S8 (Supplementary Tables S23 and S24). With the deposited per-run configuration files, every reported number is traceable to the exact configuration that produced it.

Acknowledgements and funding

T. Kadeethum, B. Badnava and V. Athreya acknowledge Siemens Energy, where part of this work was carried out as part of the authors' employment.

F. Ballarin acknowledges the INdAM GNCS Project "Metodi numerici avanzati per problemi di fluidodinamica computazionale" (CUP E53C25002010001).

H. Sethi acknowledges NVIDIA Corporation, where part of this work was carried out as part of the author's employment.

B. Chen, C. E. Heaney and C. C. Pain would like to acknowledge the following UKRI grants: **AI4Urban-Health**, "AI Solutions to Urban Health Using a Place-Based Approach" (APP55547 / UKRI 1241); and **GP4Streets**, "Pioneering Climate Adaptation in Urban Environments" (APP44894 / UKRI 1281).

Author contributions

Contributions are stated using the CRediT taxonomy.

Teeratorn Kadeethum: conceptualization; methodology; software; validation; formal analysis; investigation; resources; data curation; writing – original draft; writing – review & editing; visualization; supervision.

Babak Badnava: software; validation; formal analysis; writing – original draft; writing – review & editing.

Vignesh Athreya: resources; project administration; funding acquisition.

Boyang Chen: methodology; software; validation; formal analysis; writing – original draft; writing – review & editing.

Claire E. Heaney: methodology; software; validation; formal analysis; writing – original draft; writing – review & editing; supervision.

Francesco Ballarin: methodology; software; validation; formal analysis; investigation; resources; data curation; writing – original draft; writing – review & editing.

Harpreet Sethi: investigation; data curation; writing – original draft; writing – review & editing.

Hongying Li: investigation; data curation; writing – original draft; writing – review & editing.

Christopher C. Pain: methodology; software; validation; formal analysis; writing – original draft; writing – review & editing; supervision; project administration; funding acquisition.

Competing interests

The authors declare no competing interests.

References

[1] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation, Washington, DC, 1980.

[2] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, 12(6):620–631, 1998. doi: 10.1063/1.168744.

[3] Igor A. Baratta, Joseph P. Dean, Jørgen S. Dokken, Michal Habera, Jack S. Hale, Chris N. Richardson, Marie E. Rognes, Matthew W. Scroggs, Nathan Sime, and Garth N. Wells. DOLFINx: The next generation FEniCS problem solving environment. Preprint, Zenodo, 2023. doi: 10.5281/zenodo.10447666.

[4] Martin S. Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E. Rognes, and Garth N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100): 9–23, 2015. doi: 10.11588/ans.2015.100.20553.

[5] Federico Piscaglia and Federico Ghioldi. GPU acceleration of CFD simulations in OpenFOAM. *Aerospace*, 10(9):792, 2023. doi: 10.3390/aerospace10090792.

[6] Benjamin Alexander Pachev. *Automated GPU acceleration of stabilized shallow water solvers with FEniCSx*. PhD thesis, The University of Texas at Austin, 2024. doi: 10.26153/tsw/56005.

[7] Alberto Cattaneo, M Keith Ballard, Robert M Kirby, and Varun Shankar. JetSCI: A hybrid JAX-PETSc framework for scalable differentiable simulation. Preprint, `arXiv:2604.22087`, 2026. doi: 10.48550/arXiv.2604.22087.

[8] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows. *Computer Physics Communications*, 282:108527, 2023. doi: 10.1016/j.cpc.2022.108527.

[9] Philipp Holl and Nils Thuerey. Φ_{Flow} (PhiFlow): Differentiable simulations for PyTorch, TensorFlow and Jax. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 18515–18546. PMLR, 2024. https://proceedings.mlr.press/v235/holl24a.html.

[10] Ruiyue Huang, Claire E. Heaney, and Maarten van Reeuwijk. Parameter estimation for land-surface models using Neural Physics. *Geoscientific Model Development*, 2026. Accepted 29 June 2026. Preprint `arXiv:2505.02979`, doi: 10.48550/arXiv.2505.02979.

[11] Yu Jiao, Han Zhang, Tingkai Xue, Boyang Chen, Juntao Yang, Te Ba, Simon See, Chang Wei Kang, Claire E. Heaney, Christopher C. Pain, and Hongying Li. NP4Buoyancy: An end-to-end differentiable neural-physics framework for buoyancy-driven flows. *Journal of Computational Physics*, 564:115179, 2026. ISSN 0021-9991. doi: 10.1016/j.jcp.2026.115179.

[12] Shizheng Wen, Mingyuan Chi, Tianwei Yu, Ben Moseley, Mike Yan Michelis, Pu Ren, Hao Sun, and Siddhartha Mishra. Learning, solving and optimizing PDEs with TensorGalerkin: an efficient high-performance Galerkin assembly algorithm. Preprint, `arXiv:2602.05052`, 2026. doi: 10.48550/arXiv.2602.05052.

[13] Boyang Chen, Claire E. Heaney, and Christopher C. Pain. Neural Physics: Using AI libraries to develop physics-based solvers for incompressible computational fluid dynamics. *Computers & Fluids*, 308:106981, 2026. doi: 10.1016/j.compfluid.2026.106981. Preprint: https://arxiv.org/abs/2402.17913. Software: https://github.com/ImperialCollegeLondon/AI4PDEs.

[14] Boyang Chen, Claire E. Heaney, Jefferson L. M. A. Gomes, Omar K. Matar, and Christopher C. Pain. Solving the discretised multiphase flow equations with interface capturing on structured grids using machine learning libraries. *Computer Methods in Applied Mechanics and Engineering*, 426: 116974, 2024. doi: 10.1016/j.cma.2024.116974.

[15] Boyang Chen, Amin Nadimy, Claire E. Heaney, Mohammad Kazem Sharifian, Lluis Via Estrem, Ludovico Nicotina, Arno Hilberts, and Christopher C. Pain. Solving the discretised shallow water equations using neural networks. *Advances in Water Resources*, 197:104903, 2025. doi: 10.1016/j.advwatres.2025.104903.

[16] Toby R. F. Phillips, Claire E. Heaney, Boyang Chen, Andrew G. Buchan, and Christopher C. Pain. Solving the discretised neutron diffusion equations using neural networks. *International Journal for Numerical Methods in Engineering*, 124 (21):4659–4686, 2023. doi: 10.1002/nme.7321.

[17] Toby R. F. Phillips, Claire E. Heaney, Boyang Chen, Andrew G. Buchan, and Christopher C. Pain. Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport. Preprint, arXiv:2301.09991, 2023. doi: 10.48550/arXiv.2301.09991.

[18] Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science. *Computer Physics Communications*, 291:108802, 2023. doi: 10.1016/j.cpc.2023.108802.

[19] Leisheng Li, Ximeng Fu, Xiyu Zheng, Huiyuan Li, and Jinxi Li. GPU-accelerated finite-element method for the three-dimensional unstructured mesh atmospheric dynamic framework. *Geoscientific Model Development*, 19(15):7525–7544, 2026. doi: 10.5194/gmd-19-7525-2026.

[20] Pan Du, Yongqi Li, Mingqi Xu, and Jian-Xun Wang. DiFVM: A vectorized graph-based finite volume solver for differentiable CFD on unstructured meshes. Preprint, arXiv:2603.15920, 2026. doi: 10.48550/arXiv.2603.15920.

[21] Linfeng Li, Jiansheng Xiang, Boyang Chen, Claire E. Heaney, Steven Dargaville, and Christopher C. Pain. Implementing the discontinuous-Galerkin finite element

method using graph neural networks with application to diffusion equations. *Neural Networks*, 185:107061, 2025. doi: 10.1016/j.neunet.2024.107061.

[22] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021. doi: 10.1016/j.camwa.2020.06.009.

[23] Paul Fischer, Stefan Kerkemeier, Misun Min, Yu-Hsiang Lan, Malachi Phillips, Thilina Rathnayake, Elia Merzari, Ananias Tomboulides, Ali Karakus, Noel Chalmers, and Tim Warburton. NekRS, a GPU-accelerated spectral element Navier–Stokes solver. *Parallel Computing*, 114:102982, 2022. doi: 10.1016/j.parco.2022.102982.

[24] Martin Kronbichler and Katharina Kormann. A generic interface for parallel cell-based finite element operator application. *Computers & Fluids*, 63:135–147, 2012. doi: 10.1016/j.compfluid.2012.04.012.

[25] Jed Brown, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, Tzanio Kolev, David Medina, Will Pazner, Thilina Ratnayaka, Jeremy Thompson, and Stanimire Tomov. libCEED: Fast algebra for high-order element-based discretizations. *Journal of Open Source Software*, 6(63):2945, 2021. doi: 10.21105/joss.02945.

[26] C. Taylor and P. Hood. A numerical solution of the Navier–Stokes equations using the finite element technique. *Computers & Fluids*, 1(1):73–100, 1973. doi: 10.1016/0045-7930(73)90027-3.

[27] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3): 179–196, 1996. doi: 10.1007/BF02238511.

[28] Maxim Naumov, Marat Arsaev, Patrice Castonguay, Jonathan Cohen, Julien Demouth, Joe Eaton, Simon Layton, Nikolay Markovskiy, Istvan Reguly, Nikolai Sakharnykh, Vartika Sellappan, and Robert Strzodka. AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 37(5):S602–S626, 2015. doi: 10.1137/140980260.

[29] Hartwig Anzt, Terry Cojean, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, Yuhsiang M. Tsai, and Enrique S. Quintana-Ortí.

Ginkgo: A modern linear operator algebra framework for high performance computing. *ACM Transactions on Mathematical Software*, 48(1):2:1–2:33, 2022. doi: 10.1145/3480935.

[30] Nathan Bell, Steven Dalton, and Luke N. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing*, 34(4): C123–C152, 2012. doi: 10.1137/110838844.

[31] Richard Tran Mills, Mark F. Adams, Satish Balay, Jed Brown, Alp Dener, Matthew Knepley, Scott E. Kruger, Hannah Morgan, Todd Munson, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Junchao Zhang. Toward performance-portable PETSc for GPU-based exascale systems. *Parallel Computing*, 108:102831, 2021. doi: 10.1016/j.parco.2021.102831.

[32] Van Emden Henson and Ulrike Meier Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1): 155–177, 2002. doi: 10.1016/S0168-9274(01)00115-5.

[33] Nathan Bell, Luke N. Olson, Jacob Schroder, and Ben Southworth. PyAMG: Algebraic multigrid solvers in Python. *Journal of Open Source Software*, 8(87): 5495, 2023. doi: 10.21105/joss.05495.

[34] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2):355–365, 1987. doi: 10.1137/0724027.

[35] Clark R. Dohrmann. A preconditioner for substructuring based on constrained energy minimization. *SIAM Journal on Scientific Computing*, 25(1):246–258, 2003. doi: 10.1137/S1064827502412887.

[36] Nicole Spillane, Victorita Dolean, Patrice Hauret, Frédéric Nataf, Clemens Pechstein, and Robert Scheichl. Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps. *Numerische Mathematik*, 126(4): 741–770, 2014. doi: 10.1007/s00211-013-0576-y.

[37] Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021. doi: 10.1109/MSP.2020.3016905.

[38] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid PDE solvers. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *PMLR*, pages 2415–2423, 2019.

[39] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *PMLR*, pages 6489–6499, 2020.

[40] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *PMLR*, pages 8459–8468, 2020.

[41] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations (ICLR)*, 2021. Preprint `arXiv:2010.03409`, doi: 10.48550/arXiv.2010.03409.

[42] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations (ICLR)*, 2022. Preprint `arXiv:2202.03376`, doi: 10.48550/arXiv.2202.03376.

[43] The OpenFOAM Foundation. OpenFOAM version 12. https://openfoam.org/version/12/, 2024. Accessed 26 August 2026.

[44] G. de Vahl Davis. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3): 249–264, 1983. doi: 10.1002/fld.1650030305.

[45] P. Le Quéré. Accurate solutions to the square thermally driven cavity at high Rayleigh number. *Computers & Fluids*, 20(1):29–41, 1991. doi: 10.1016/0045-7930(91)90025-D.

[46] Hans Johnston and Charles R. Doering. Comparison of turbulent thermal convection between conditions of constant temperature and constant flux. *Physical Review Letters*, 102(6):064501, 2009. doi: 10.1103/PhysRevLett.102.064501.

[47] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit-explicit Runge–Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25(2–3):151–167, 1997. doi: 10.1016/S0168-9274(97)00056-1.

[48] M. Khatamifar, W. Lin, and L. Dong. Transient conjugate natural convection heat transfer in a differentially-heated square cavity with a partition of finite thickness and thermal conductivity. *Case Studies in Thermal Engineering*, 25:100952, 2021. doi: 10.1016/j.csite.2021.100952.

[49] J. M. House, C. Beckermann, and T. F. Smith. Effect of a centered conducting body on natural convection heat transfer in an enclosure. *Numerical Heat Transfer, Part A: Applications*, 18(2):213–225, 1990. doi: 10.1080/10407789008944791.

[50] ANSYS, Inc. Conjugate heat transfer simulation of a single-cylinder internal combustion engine. In *Ansys Forte Tutorials*, Release 2025 R1, chapter 24, 2025. Accessed via the ANSYS Help system, 26 August 2026.

[51] J. B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill Education, New York, 2nd edition, 2018. Ch. 12, heat transfer in engines.

[52] T. Fusegi, J. M. Hyun, K. Kuwahara, and B. Farouk. A numerical study of three-dimensional natural convection in a differentially heated cubical enclosure. *International Journal of Heat and Mass Transfer*, 34(6):1543–1557, 1991. doi: 10.1016/0017-9310(91)90295-P.

[53] Joe Alexandersen, Ole Sigmund, and Niels Aage. Large scale three-dimensional topology optimisation of heat sinks cooled by natural convection. *International Journal of Heat and Mass Transfer*, 100:876–891, 2016. doi: 10.1016/j.ijheatmasstransfer.2016.05.013.

[54] J. L. Guermond, P. Minev, and Jie Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195(44–47):6011–6045, 2006. doi: 10.1016/j.cma.2005.10.010.

[55] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi: 10.1002/nme.2579.

[56] Satish Balay, Shrirang Abhyankar, Mark F. Adams, et al. PETSc/TAO users manual. Argonne National Laboratory, ANL-21/39, Revision 3.25, 2026. doi: 10.2172/3025790. https://petsc.org/.

[57] Philippe R. Spalart, Robert D. Moser, and Michael M. Rogers. Spectral methods for the Navier–Stokes equations with one infinite and two periodic directions. *Journal of Computational Physics*, 96(2):297–324, 1991. doi: 10.1016/0021-9991(91)90238-G.

[58] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, et al. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and*

Operating Systems (ASPLOS '24), volume 2, pages 929–947, 2024. doi: 10.1145/3620665.3640366.

[59] E. Tric, G. Labrosse, and M. Betrouni. A first incursion into the 3D structure of natural convection of air in a differentially heated cubic cavity, from accurate numerical solutions. *International Journal of Heat and Mass Transfer*, 43(21): 4043–4056, 2000. doi: 10.1016/S0017-9310(00)00037-5.

Table 1. Verification and validation summary. The correctness case in one place, in the order it is established: the solver solves the stated equations, the reported solutions are grid-converged, and only then are they compared against accepted references. Deviations are relative unless stated. On the conjugate cavity we report agreement against the range of prior independent solutions rather than against a single study, because our value lies inside that range in all six cells while Khatamifar et al. [48]'s own value lies outside it in three of the six; the 0.3–2.9% column is the deviation from their value. Per-mesh and per-solver detail is in Extended Data Table 1 (cavity), Extended Data Table 2 (conjugate) and Extended Data Table 4 (manufactured solutions). The per-mesh Nusselt values behind the grid-convergence rows, the RK3–CN and ARS(2,2,2) comparison, the extrapolated values and the GCI inputs are in Supplementary Table S10; the decomposition rows are reported in Supplementary Note S7.

Test	Reference	This work	Agreement
Code verification: does the solver solve the equations it states?			
Manufactured solution, fully 3D Beltrami	Taylor–Hood orders $\{3, 3, 2\}$	2.95 (u, v, w), 2.94 (T), 2.03 (p)	≤ 0.06 in order
Conjugate manufactured solution	$\mathcal{O}(h^3)$ in temperature	attained	—
Two-material conduction slab	analytic $q^*/[2(k_s{+}k_f)]$	reproduced	$< 10^{-13}$
Solution verification: is the reported solution converged, and independent of the march?			
Grid convergence, conjugate Ra $= 10^6$	five-mesh $\sqrt{2}$ sequence, RK3–CN	observed order 2.92	GCI 0.011%
Grid convergence, conjugate Ra $= 10^5$	five-mesh $\sqrt{2}$ sequence, RK3–CN	observed order 3.7, tolerance-limited	GCI 0.001%
Grid convergence, conjugate Ra $= 10^4$	five-mesh $\sqrt{2}$ sequence, RK3–CN	order not resolved (near-conduction)	grid spread 0.03%
Scheme robustness	ARS(2,2,2) vs RK3–CN	same five meshes	≥ 5 sig. figs
Decomposition, Rayleigh–Bénard	single-GPU trajectory	1 versus 2 GPUs, 120 steps	8×10^{-11} in Nu
Decomposition, conjugate	single-GPU reference run, Nu $= 2.7995$	1 and 2 GPUs	1.4×10^{-7}
Validation: does it match accepted references?			
Cavity, 2D, Ra $\leq 10^5$	de Vahl Davis [44]	finest mesh	$\leq 0.09\%$
Cavity, 2D, Ra $= 10^6$	Le Quéré [45], spectral 8.8252	Nu $= 8.8257$	0.005%
Cavity, 3D, Ra $= 10^3$–10^5	Tric et al. [59], pseudo-spectral	finest mesh	$\leq 0.11\%$
Conjugate cavity, six cases	Khatamifar et al. [48] and the prior-study range	inside the published range in 6/6	0.3–2.9%
Centred conducting body	House et al. [49]	four of five cases	$\leq 0.23\%$ (4/5 cases); 1.6% (5th)
Turbulent Rayleigh–Bénard, Ra $= 10^7$	Johnston and Doering [46] fit	$\langle \mathrm{Nu} \rangle \approx 14.0$	$\approx 3\%$

Table 2. Every matched same-discretization comparison in this work (Tier B, Section 4.2). Four problems, two and three dimensional, spanning nearly three decades of problem size. In all of them the discretization, mesh, time step, step count, pressure tolerance (10^{-6}) and mass/diffusion tolerance (10^{-10}) are held fixed, and only the linear-algebra backend and the execution device change: graph multigrid on one H200 against PETSc on 32 CPU cores. Ranges run from the coarsest to the finest mesh of each sequence. The ratio in the last column is the CPU time per step divided by the GPU time per step, so a value above one favours the GPU. It is not a property of the hardware pair. It is a function of problem size, rising from below parity on the smallest engine mesh to 181× on the largest heat-sink mesh. Two qualifications carry throughout. The engine and heat-sink GPU drivers run eagerly; CUDA-graph capture was not evaluated for those cases. The whole-step figures are not attributable to the multigrid alone; the pressure solve contributes 18.7× of the largest of them (Table 3). Per-problem tables are Supplementary Tables S11 to S14.

Problem	Meshes	Coupled DOF	GPU (ms)	CPU (ms)	CPU/GPU
Conjugate cavity (2D)	6	3.9×10^4–1.2×10^6	40.5–60.8	50.7–1452.9	1.25–23.9
Turbulent Rayleigh–Bénard (2D)	2	1.1×10^5–4.3×10^5	76.8–171.2	167.3–613.8	2.2–3.6
Engine, body-fitted (3D)	3	3.6×10^4–7.4×10^5	161.8–241.6	130.9–5771.0	0.81–23.9
Conjugate heat sink (3D)	3	3.9×10^5–1.2×10^7	228.9–1598.5	3302.2–289614.5	14.4–181.2

Table 3. Per-phase timings for the m3 conjugate heat-sink case, excluding the first step. Percentages use the measured total step time; time not assigned to an instrumented phase is reported separately. That remainder (control transfers, synchronization, allocation and launch overhead) is 28.5% of the GPU step and 3.5% of the CPU step. The whole-step CPU/GPU ratio is 181, compared with 18.7 for the pressure solve, which is 6.8% of the GPU step and 0.7% of the CPU step. The pressure solve therefore has the lowest ratio of any phase, so removing it from both implementations would raise the whole-step ratio rather than lower it. The pressure tolerance is 10^{-6} in both implementations and is independent of the mass tolerance.

Phase	GPU (ms)	CPU (ms)	ratio	% of CPU step
implicit temperature	340.6	114 594.5	336	39.6
implicit velocity	227.4	87 678.2	386	30.3
projection corrector	140.6	40 593.9	289	14.0
advection assembly	246.8	21 897.1	89	7.6
other (spmv, BCs)	79.0	12 587.4	159	4.3
pressure solve	108.9	2033.2	**18.7**	**0.7**
unattributed remainder	455.2	10 230.2	–	3.5
measured step	**1598.5**	**289 614.5**	**181**	**100.0**

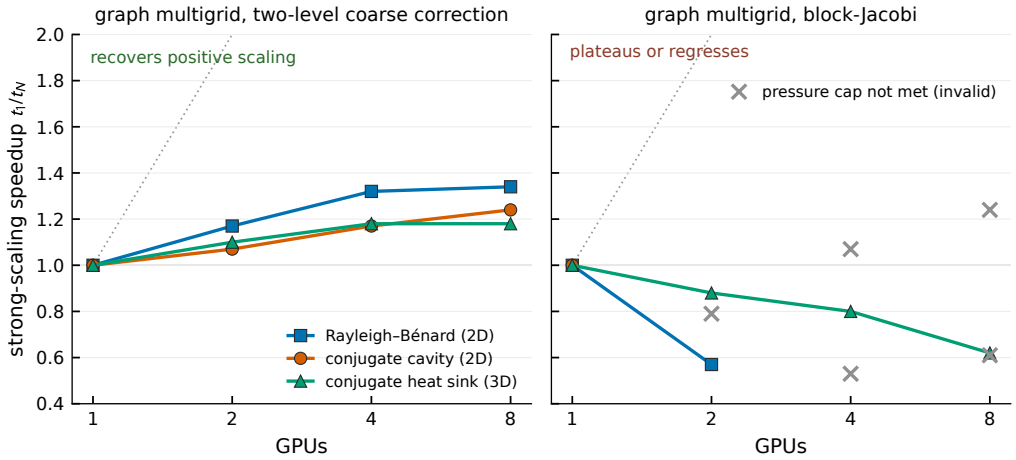
Extended Data

Extended Data Table 1. Converged Nu on the 2D Boussinesq cavity benchmark. The two P_2–P_1 solvers (GNN, FEniCSx) and the cell-centred OpenFOAM v12 (OF12) agree to $\lesssim 1\%$ on the two finer meshes; the GNN and FEniCSx share the *same* Taylor–Hood discretization, and the cell-centred OF12 overshoots on the coarsest mesh (to 2.4% at $\mathrm{Ra} = 10^6$, h_{64}). The 0.35% split at $\mathrm{Ra} = 10^4$, h_{256} is *not* a discretization or operator difference: three of the four solvers agree there (CNN, GNN and OF12 all give 2.245) and the FEniCSx value (2.237) is the lone outlier. The two P_2–P_1 codes assemble bit-identical operators from the same UFL forms, the GPU advection matrix reproducing the compiled finite element form to $\sim 10^{-14}$, and share the wall-Nusselt functional; the split shrinks to $\approx 0.05\%$ at h_{128}. We report the cell as an unresolved outlier. The suspected cause is that the FEniCSx h_{256} solve did not reach steady state at the finest, slowest-converging mesh, but this has not been demonstrated. The cell is excluded from the refinement trend (Supplementary Note S8.5).

Ra	mesh	CNN	GNN	FEniCSx	OF12
10^4	h_{64}	2.250	2.245	2.245	2.249
	h_{128}	2.246	2.245	2.244	2.246
	h_{256}	2.245	2.245	2.237	2.245
10^5	h_{64}	4.560	4.524	4.523	4.556
	h_{128}	4.531	4.522	4.522	4.530
	h_{256}	4.524	4.519	4.519	4.523
10^6	h_{64}	9.088	8.846	8.846	9.055
	h_{128}	8.884	8.828	8.828	8.883
	h_{256}	8.840	8.826	8.826	8.839

Extended Data Table 2. Validation of the conjugate cavity Nusselt number against Khatamifar et al. [48]. Differentially-heated square cavity with a central vertical partition, $T_p = 0.1$, $X_p = 0.5$, $\mathrm{Pr} = 0.71$. "This work" is the grid-converged GNN value over the five-mesh sequence; "Khatamifar (2021)" is their finite volume result; "prior studies" is the range of the independent numerical solutions compiled in their Table 3. The GNN values fall within the published inter-study spread at every Rayleigh number and conductivity ratio.

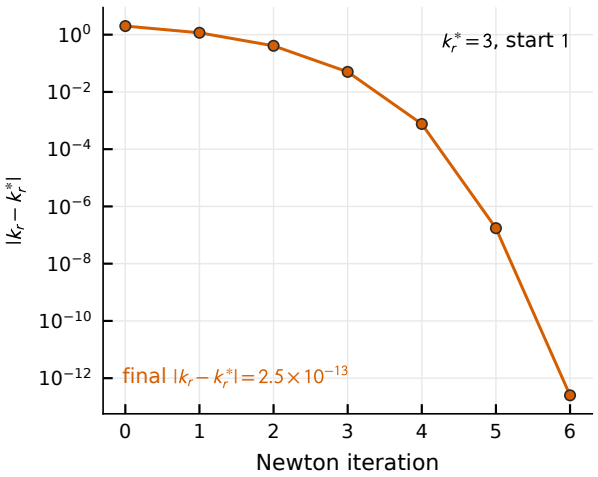
k_r	Source	$\mathrm{Ra} = 10^4$	$\mathrm{Ra} = 10^5$	$\mathrm{Ra} = 10^6$
1	This work (GNN)	1.059	1.751	2.794
	Khatamifar (2021)	1.09	1.76	2.82
	prior studies	1.05–1.06	1.74–1.79	2.79–2.80
100	This work (GNN)	1.166	2.079	3.913
	Khatamifar (2021)	1.17	2.11	4.03
	prior studies	1.05–1.19	1.87–2.14	3.51–3.93



Extended Data Fig. 1. A two-level coarse correction recovers positive strong scaling across three buoyant and conjugate problems. Fixed-problem strong-scaling speedup t_1/t_N on H200 GPUs, a same-configuration A/B in which *only* the pressure preconditioner varies (mesh, P_2–P_1 discretization and RK3–CN integrator held identical). The QoI is insensitive at the printed precision over the sampled window, although the capped block-Jacobi points do not meet the pressure tolerance. **(Left)** A global aggregation coarse correction turns a regressing curve into a weakly positive one on all three problems, reaching 1.18–1.34× at eight GPUs (14.7–16.8% parallel efficiency), because it flattens the per-rank pressure iteration count. **(Right)** Per-rank block-Jacobi plateaus or regresses: its pressure iterations grow with rank, and where they saturate the per-solve cap the pressure is under-resolved. Those points are drawn as grey crosses and are not valid strong-scaling measurements. Fixed problem sizes: 5.5×10^7 (Rayleigh–Bénard, $\mathrm{Ra}=10^7$), 5.4×10^7 (conjugate cavity, $\mathrm{Ra}=10^6$) and 1.6×10^7 DOF (3D heat sink); the dotted line is ideal linear speedup. All four problems, including the engine, are tabulated in Supplementary Table S20.

Extended Data Table 3. The identical graph multigrid on CPU and GPU. Conjugate cavity ($\mathrm{Ra}=10^6$, $T_p=0.1$, $k_r=1$, rotational-IPCS + RK3–CN, fp64), fixed 300 steps with the steady checker off (convergence-free per-step timing from the wall-versus-step slope), median of three repetitions; per-cell spreads are below 7% except one contended CPU repetition at m_1 (12.1 s/step), which does not affect the median. The two runs use the same solver from byte-identical configurations except for two keys: the execution device, and CUDA-graph capture, which exists only on the GPU backend and is enabled there. Capture is worth roughly 2× on pressure-dominated work in this solver, so it is a material part of the ratio at the coarse meshes and a small part of it at m_6, where the CPU run is 21× slower.

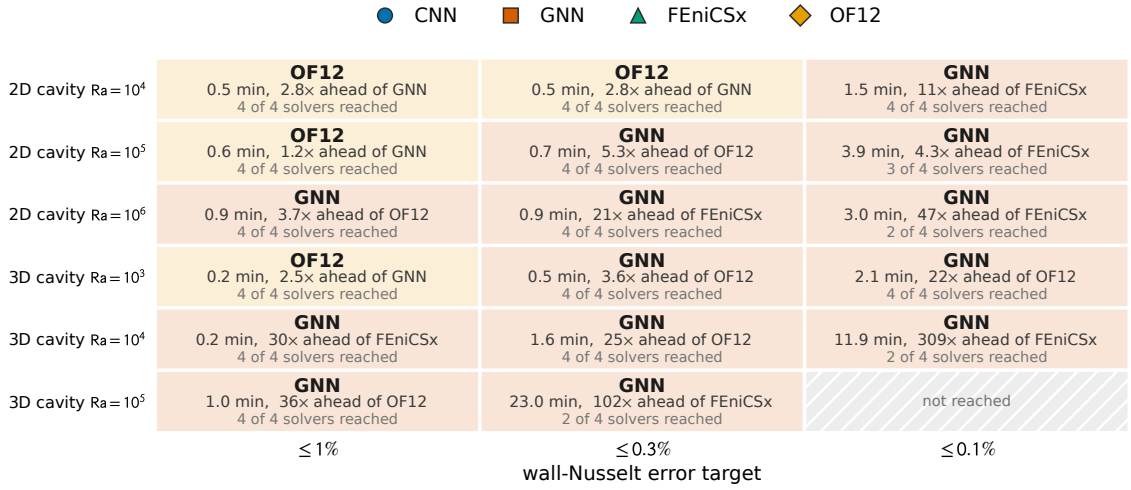
Mesh	P_2 DOF	GPU (ms/step)	CPU (ms/step)	CPU/GPU
m1	11 963	40.4	184.3	4.6
m3	46 929	42.6	248.4	5.8
m5	185 949	48.1	426.9	8.9
m6	373 073	60.5	1293.2	21.4



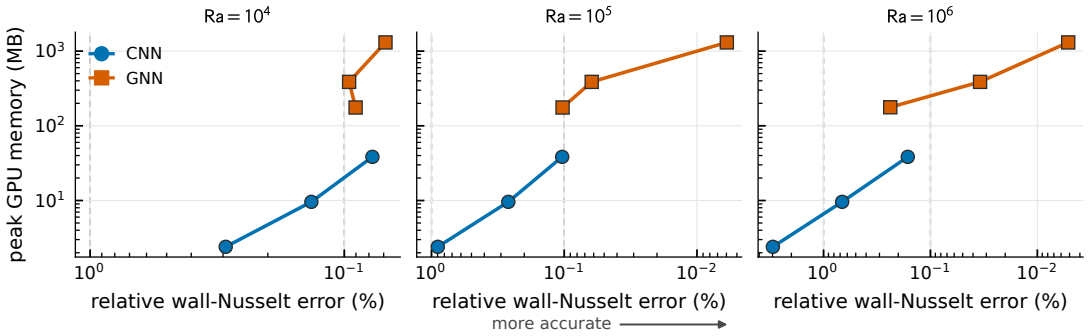
Extended Data Fig. 2. The gradient recovers an unknown conductivity ratio. A Newton iteration whose derivative comes from reverse-mode differentiation of the coupled solve, applied to the conjugate cavity at $\mathrm{Ra} = 10^3$. Starting from $k_r = 1$ against a true $k_r^* = 3$, the error falls quadratically to 2.5×10^{-13} in six iterations, each costing one coupled solve and one reverse pass. No hand-derived adjoint is used. This is a differentiability demonstration on a small inverse problem rather than an optimization study, and it is a consistency result on the forward map: the target Nusselt number is generated by the same program that is then inverted.

Extended Data Table 4. Fully three-dimensional GNN method-of-manufactured-solutions (MMS) convergence. Discrete L^2 error of the unstructured P_2–P_1 GNN–SIMPLE solver on the fully 3D Beltrami manufactured field (u, v, w all non-zero, divergence-free; $\mathrm{Ra} = 1$, $\mathrm{Pr} = 0.71$), and the mesh-pairwise convergence order $\log(\|e\|^{(n)}/\|e\|^{(2n)})/\log(\bar{h}_n/\bar{h}_{2n})$. The tabulated $\bar{h}$ (mean cell size) gives refinement ratios $\bar{h}_{16}/\bar{h}_{32} = 1.92$ and $\bar{h}_{32}/\bar{h}_{64} = 2.03$, not exactly 2 on the unstructured tetrahedralization, so the orders are taken against the actual, not nominal, spacing. The out-of-plane component w converges at the same Taylor–Hood rate as u and v; the $h_{32} \to h_{64}$ rates sit within 0.06 of the theoretical $\{3, 3, 2\}$.

Mesh	$\bar{h}$	e_u	e_v	e_w	e_T	e_p
h_{16}	0.042 87	3.27×10^{-4}	3.28×10^{-4}	6.24×10^{-4}	3.74×10^{-5}	2.18×10^{-2}
h_{32}	0.022 31	3.76×10^{-5}	3.76×10^{-5}	7.14×10^{-5}	4.37×10^{-6}	4.86×10^{-3}
h_{64}	0.010 98	4.62×10^{-6}	4.63×10^{-6}	8.83×10^{-6}	5.45×10^{-7}	1.15×10^{-3}
order $16 \to 32$		3.31	3.32	3.32	3.28	2.30
order $32 \to 64$		2.95	2.95	2.95	2.94	2.03

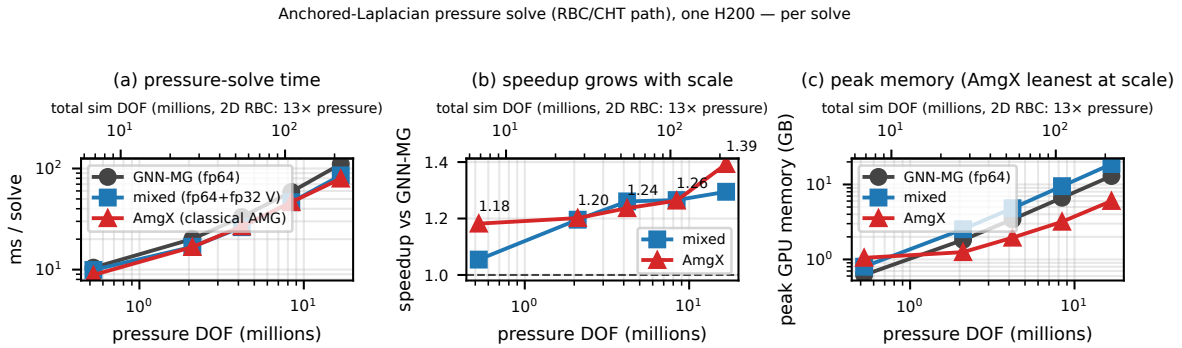


(a)

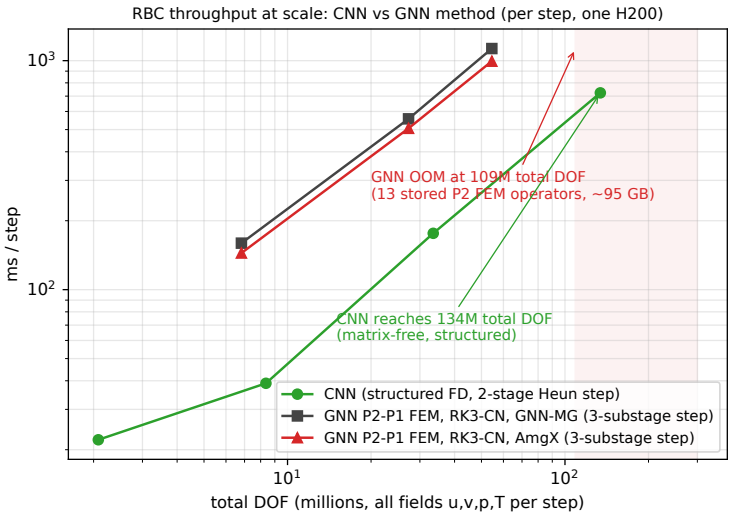


(b)

Extended Data Fig. 3. Fastest solver to a given accuracy, and peak GPU memory against achieved accuracy. Both panels index by *achieved accuracy* rather than by mesh: at matched mesh the colocated schemes carry 3.8× fewer unknowns in 2D and 5.0–5.9× in 3D (Supplementary Note S3.1), so a mesh ranking would favour them before any solver runs. **(a)** The fastest solver to a given accuracy, 2D and 3D cavity. Each cell gives the fastest solver, its wall time at its cheapest qualifying mesh, and its lead over the next-fastest solver that also reaches the target. The leader changes with the target: OpenFOAM v12 on the easy cells at loose tolerance, by as little as 1.2×, then the graph solver as the demand tightens, by 47× on the hardest 2D cell and 309× on the hardest 3D one, where the field that reaches the target at all narrows to two solvers. Hatched cells were reached by none. **(b)** Peak GPU memory against achieved accuracy, two-dimensional cavity, the two GPU solvers only, since GPU and CPU memory are not comparable resources. Axes and reference values as in Figure 3. The P_2–P_1 pair carries an order of magnitude or more peak memory than the colocated CNN at a matched mesh, and 4.6× at fixed accuracy at Ra = 10^5, where the two solvers are accuracy-matched. At fixed accuracy the graph solver is the faster of the two and the less memory-frugal, a trade of memory for time. Both panels are Tier C (Section 4.2); neither is same-method or hardware-isolated.

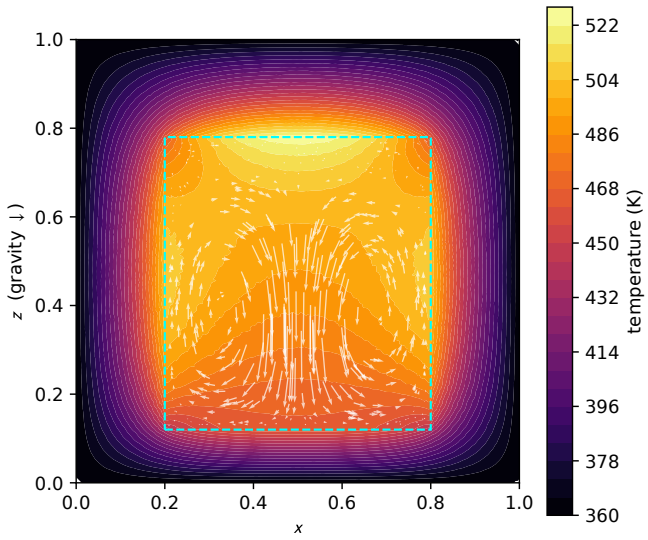


(a) Single-GPU pressure-solve throughput and peak memory vs DOF (frozen assembled Laplacian, RBC and CHT path), one H200. fp64 graph-multigrid PCG vs mixed-precision (fp64 Krylov + fp32 V-cycle) vs AmgX (classical AMG), all to 10^{-6}. **(a)** per-solve time; **(b)** speedup vs fp64 multigrid, growing with scale; **(c)** peak GPU memory (AmgX leanest, measured device-wide to capture AmgX's own allocations). Top axis: equivalent total simulation DOF (13× pressure for the 2D RBC P_2–P_1 problem). Per-solve; the per-step application gain is smaller (Supplementary Note S5.4).

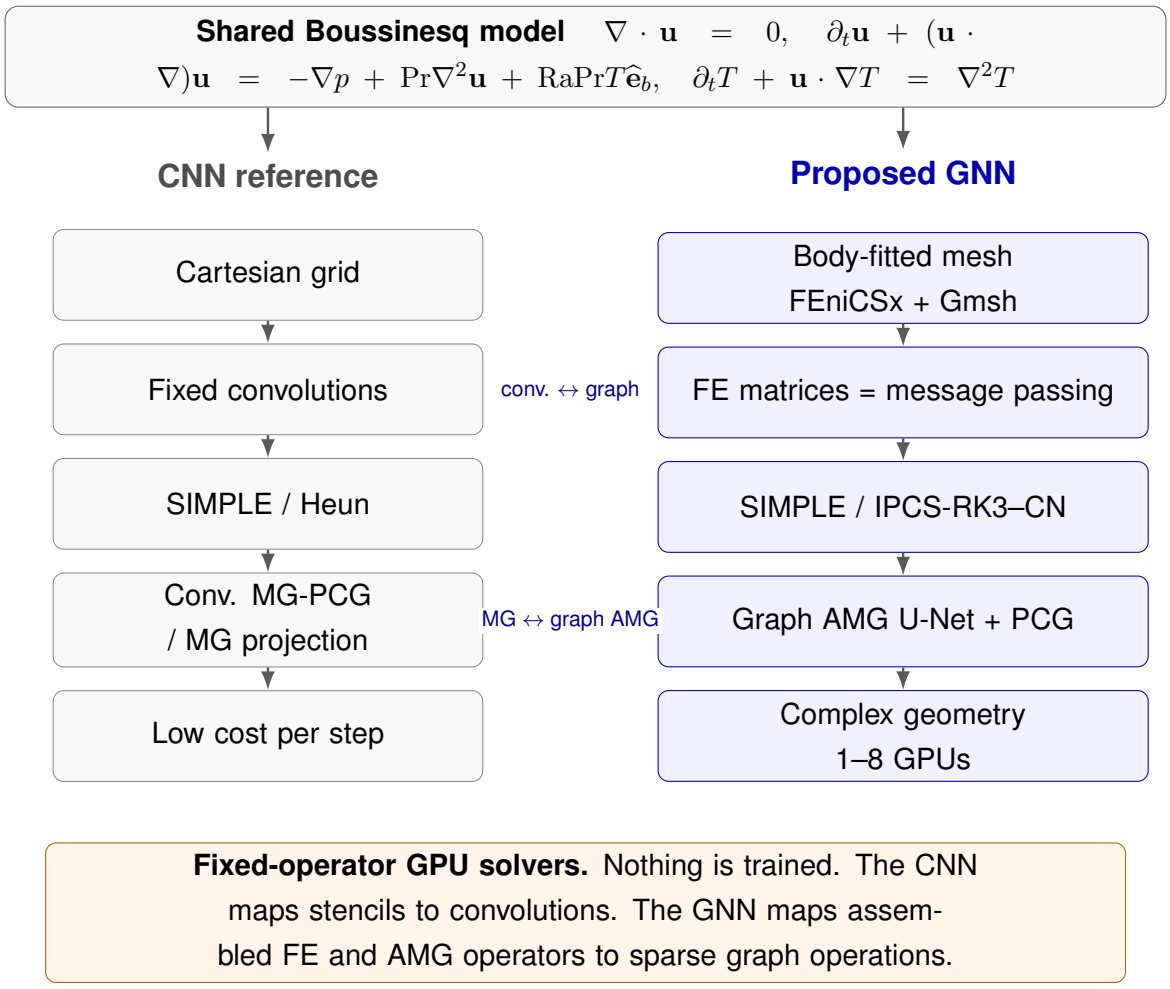


(b) Per-step wall time vs total DOF (all fields), structured CNN (finite difference, one complete two-stage explicit Heun step with a single pressure projection) vs unstructured GNN (P_2–P_1 FEM, one complete RK3–CN step of three implicit substages, each with its own pressure projection; multigrid and AmgX pressure), one H200, Ra $= 10^7$ Rayleigh–Bénard. Not like-for-like: the per-step gap is method+integrator, not solver. The CNN reaches 1.3×10^8 total DOF (matrix-free); the GNN full solver is finite element-operator-bound near 5.5×10^7. CNN sampled at power-of-two n (its geometric multigrid requirement).

Extended Data Fig. 4. Single-GPU scaling with problem size. The isolated pressure solve and then the full per-step cost, both against total degrees of freedom on one H200.



Extended Data Fig. 5. Single-cylinder engine conjugate heat transfer. Steady mid-plane temperature (K, cast-iron map): the gas bore (dashed outline) is heated by the prescribed cycle-averaged combustion flux on the chamber wall and convects under buoyancy (vectors: hot gas rises to the fire deck and falls along the cooler side walls); the surrounding metal conducts the heat to the cooled outer faces. Monolithic conjugate solve, graph-multigrid pressure, finest mesh (1.8×10^5 scalar P_2 DOF per field).



Extended Data Fig. 6. Overview of the CNN and the proposed GNN solvers. Both solvers target the same Boussinesq system. The CNN uses fixed convolutions on a Cartesian grid. The proposed GNN uses body-fitted finite elements assembled with FEniCSx and executed as sparse graph operations. Its pressure solve uses a graph-AMG U-Net preconditioner and extends naturally to complex geometry and multi-GPU execution.

Supplementary Information

Classical finite-element solvers as differentiable graph programs on GPUs

S1 Verification of the discretization and of the graph execution

Abbreviations and labels. GNN: our finite element solver, expressed as fixed graph operations; nothing is trained. CNN: the structured finite difference solver, expressed as convolutions. FEM: finite element method. SIMPLE: the steady pressure-correction iteration (Semi-Implicit Method for Pressure-Linked Equations). IPCS: incremental pressure-correction scheme. RK3–CN: third-order Runge–Kutta advection with Crank–Nicolson diffusion. ARS(2,2,2): the implicit-explicit Runge–Kutta scheme of main-text Section 2.5. MMS: method of manufactured solutions. DOF: degrees of freedom. fp64 and fp32: double and single precision. CG and PCG: conjugate gradients and preconditioned conjugate gradients. AMG: algebraic multigrid. GAMG: PETSc's geometric algebraic multigrid; OpenFOAM's solver of the same name appears only in Supplementary Note S8. RBC: Rayleigh–Bénard convection. GCI: grid convergence index. T_p: partition thickness of the conjugate cavity. A mesh label h_n names a mesh with n nominal cells per unit edge (main-text Section 4.2). The conjugate-cavity meshes are named m1 to m6; heat-sink and engine mesh names are given in each table.

Manufactured solutions show that the GNN–SIMPLE solver (Taylor–Hood P_2–P_1 finite elements) attains its theoretical orders in 3D: 3 for velocity and temperature, 2 for pressure. The CNN–SIMPLE solver (9-point Mehrstellen finite differences) reaches a velocity and pressure error floor set by its boundary closure. The graph V-cycle reproduces a reference cycle to fp64 round-off. FEniCSx and OpenFOAM v12 are not re-verified.

S1.1 Manufactured solution and forcing

We use the steady, divergence-free Taylor–Green solution on $\Omega = (0,1)^d$. In two dimensions,

$$\boldsymbol{u}(\boldsymbol{x}) = \begin{pmatrix} \sin(2\pi x)\,\cos(2\pi y) \\ -\cos(2\pi x)\,\sin(2\pi y) \end{pmatrix}, \quad T(\boldsymbol{x}) = \sin(\pi x)\,\sin(\pi y), \quad p(\boldsymbol{x}) = \cos(2\pi x)\,\cos(2\pi y). \tag{S1}$$

In three dimensions both GPU solvers use the fully three-dimensional Beltrami-type field $u = s_{2x}c_{2y}c_{2z}$, $v = c_{2x}s_{2y}c_{2z}$, $w = -2c_{2x}c_{2y}s_{2z}$ (divergence-free), with $T = s_{\pi x}s_{\pi y}s_{\pi z}$ and $p = c_{2x}c_{2y}c_{2z}$. Here $s_{2x} = \sin(2\pi x)$, $c_{2x} = \cos(2\pi x)$ and $s_{\pi x} = \sin(\pi x)$, and likewise in y

and z. All three velocity components are non-zero, so the forcing drives all three momentum equations. The momentum and energy sources $\boldsymbol{f}$ and q are computed analytically and injected as right-hand sides of the SIMPLE iteration.

Both solvers impose inhomogeneous Dirichlet data from the manufactured solution. The CNN uses ghost-cell extrapolation $\phi_{\text{ghost}} = 2\,\phi_{\text{exact}}(\boldsymbol{x}_{\text{wall}}) - \phi_{\text{interior}}$, with ϕ_{exact} evaluated at the wall coordinate so that the interpolated wall value is exact. The GNN replaces the rows of the boundary P_2 nodes, and the solved boundary values equal the imposed ones bit for bit.

Each mesh is run to a pseudo-steady fixed point, except the fully three-dimensional h_{16} mesh, where the velocity update does not settle and cycles at the outer-iteration cap. The error of the discrete field ϕ_h, $\phi \in \{u, v, w, p, T\}$, is measured in the same FEM mass-weighted norm for both solvers,

$$\|e_\phi\|_{L^2} \;=\; \Big(\int_\Omega (\phi_h - \phi)^2 \,\mathrm{d}x\Big)^{1/2}. \tag{S2}$$

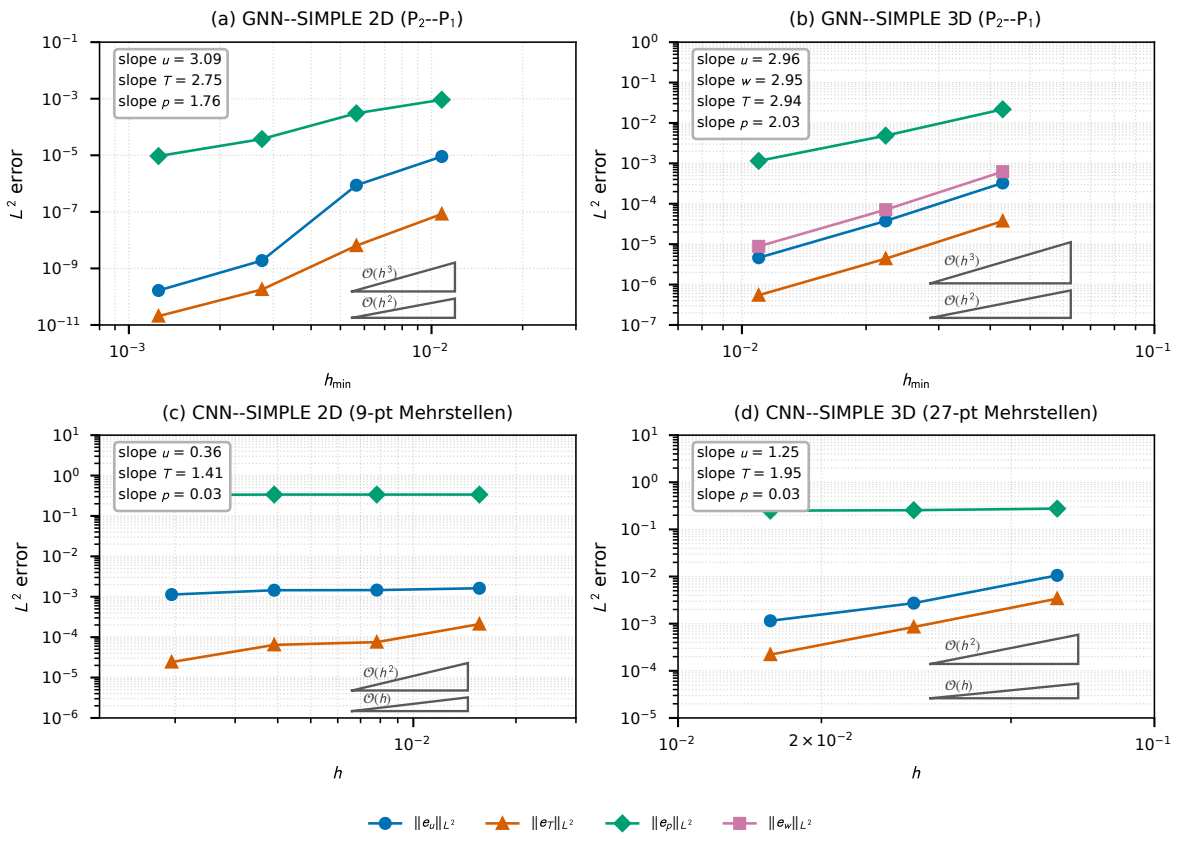
The order between adjacent meshes with $h_1 > h_2$ is $\log\big(\|e_\phi\|^{(h_1)}/\|e_\phi\|^{(h_2)}\big)\big/\log(h_1/h_2)$. It is taken with the realized mesh ratio, which is not exactly 2 on the unstructured sequences, except in Supplementary Table S1 and in the 2D block of Supplementary Table S2, where the orders use the nominal ratio 2.

S1.2 Empirical convergence rates

GNN–SIMPLE (panels a, b). On the fully three-dimensional field the orders between h_{32} and h_{64} lie within 0.06 of the theoretical $\{3, 3, 2\}$. They are ≈ 2.95 for all three velocity components, 2.94 for temperature and 2.03 for pressure (Extended Data Table 4). The slopes between the two finest 2D meshes in Supplementary Figure S1a, taken against the minimum mesh spacing, are lower, ≈ 2.75 for temperature and ≈ 1.76 for pressure. Two effects lower them. The minimum spacing shrinks by more than a factor of two between h_{256} and h_{512}, and there the velocity error falls below $\sim 2 \times 10^{-9}$, near the residual floor set by the iteration tolerance.

GNN operator-level verification. Three operator checks on the GNN pass at machine precision or at the theoretical rate. The assembled divergence and gradient are exact transposes: $\langle B_x u, p\rangle = -\langle G_x^T u, p\rangle$ to a relative $\lesssim 10^{-15}$ for random u vanishing on $\partial\Omega$. A pure pressure Poisson solve with the SIMPLE Laplacian and mass matrix converges at orders 2.13 and 1.93. A pure Stokes solve with the SIMPLE Schur-complement PCG gives orders of 3.1 to 3.9 in velocity, above the formal 3, and ≈ 2 in pressure.

Multi-GPU GNN MMS at scaling-study sizes. On four GPUs the distributed solver converges at order 3.04 in velocity and 2.04 in pressure between h_{64} and h_{128} (Sup-



Supplementary Figure S1. MMS convergence for the two GPU-native SIMPLE solvers in 2D and 3D. Discrete L^2 error against mesh resolution, with one legend for all panels and e_w in panel (b) only. **(a, b)** GNN–SIMPLE, with the pairwise 3D orders in Extended Data Table 4. **(c, d)** CNN–SIMPLE, with ghost-cell Dirichlet data evaluated at the wall coordinate. Legend slopes are the slopes between the two finest meshes. The abscissa is $h_{\min}$ on the unstructured meshes and h on the Cartesian grid.

plementary Table S1). These runs use the z-dependent, w-free variant of Supplementary Equation (S1), with each field carrying a $\cos(2\pi z)$ or $\sin(\pi z)$ factor, and with Nicolaides deflation or block-Jacobi as the pressure preconditioner (Supplementary Note S7). The one- and four-GPU velocity errors agree to a relative $\sim 10^{-9}$ up to h_{64}. The two preconditioners agree to 1.6×10^{-9} in velocity and 2.2×10^{-8} in pressure up to h_{64}, and to $\sim 6 \times 10^{-7}$ at h_{128}, where their iterative pressure floors differ.

Supplementary Table S1. Multi-GPU MMS. GNN–SIMPLE errors and orders on four GPUs; "ord" is the order between the current row and the row above. The Nicolaides deflation and block-Jacobi preconditioners give the same values to every digit shown.

Mesh	$\|e_u\|_{L^2}$	ord$_u$	$\|e_T\|_{L^2}$	ord$_T$	$\|e_p\|_{L^2}$	ord$_p$
h_{16}	3.07×10^{-4}	–	3.73×10^{-5}	–	1.95×10^{-2}	–
h_{32}	3.57×10^{-5}	3.10	4.38×10^{-6}	3.09	4.55×10^{-3}	2.10
h_{64}	4.41×10^{-6}	3.02	5.44×10^{-7}	3.01	1.10×10^{-3}	2.05
h_{128}	5.37×10^{-7}	3.04	6.62×10^{-8}	3.04	2.66×10^{-4}	2.04

CNN–SIMPLE (panels c, d): boundary-stencil floor. In 2D the CNN velocity and pressure errors stall at $\sim 1.5 \times 10^{-3}$ and $\sim 3.4 \times 10^{-1}$ for h_{256} and coarser. A linear ramp in y carries over 98% of $\|e_p\|_{L^2}^2$ at every mesh. The exact velocity has a discrete divergence of $\sim 10^{-14}$, so the error enters through momentum. Boundary cells lose order in the Mehrstellen Laplacian and the ghost-cell pressure gradient. SIMPLE absorbs this truncation as the pressure ramp, which forces the v-momentum equation everywhere. Temperature is uncoupled from the pressure correction and converges at $\mathcal{O}(h^{1.5})$.

S1.3 Conjugate heat transfer: variable-conductivity MMS

A manufactured solution with a tenfold conductivity jump verifies the GNN conjugate energy operator to third order in temperature. The domain $\Omega = (0,1)^d$ splits along the facet-aligned plane $y_\Gamma = \frac{1}{2}$ into a solid ($y \le y_\Gamma$, $k_s = 0.1$) and a fluid ($y > y_\Gamma$, $k_f = 1$). The energy equation is solved monolithically with $\boldsymbol{u} \equiv \mathbf{0}$ everywhere, so the test is conduction only. It verifies the variable-conductivity conduction operator on a facet-aligned interface whose kink the P_2 space represents exactly, not convection across the interface. The C^0 space enforces temperature continuity across the interface $\Gamma = \{y = y_\Gamma\}$, and the natural boundary term of the weak form enforces flux continuity.

Manufactured solution with a conductivity kink. A conjugate solution needs continuous T and continuous $k\,\partial_n T$, so $\partial_n T$ must kink where k jumps. With $\eta = y - y_\Gamma$ and tangential shape $g = \sin(\pi x)$ in 2D ($g = \sin(\pi x)\sin(\pi z)$ in 3D),

$$T(\boldsymbol{x}) = \begin{cases} \left[\, a + b\,\eta + c_s\,\eta^2 \,\right] g, & y \le y_\Gamma \quad \text{(solid)}, \\ \left[\, a + b\,(k_s/k_f)\,\eta + c_f\,\eta^2 \,\right] g, & y > y_\Gamma \quad \text{(fluid)}, \end{cases} \tag{S3}$$

with $(a, b, c_s, c_f) = (0.4,\ 0.7,\ 0.35,\ -0.2)$. Both branches equal $a\,g$ at $\eta = 0$, and $k_s\,\partial_y T_s = k_s b\,g = k_f\,\partial_y T_f$. The volumetric forcing $q = -k\,\nabla^2 T$ in each region is computed in closed form and injected as the energy right-hand side.

Manufactured-solution self-consistency. Automatic differentiation confirms that the injected source equals $-\nabla\cdot(k\nabla T)$ for the exact field of Supplementary Equation (S3) to a relative error $\lesssim 5 \times 10^{-16}$ in both subdomains, in 2D and 3D.

Convergence. The P_2 temperature error converges at $\approx \mathcal{O}(h^3)$ or faster in the fluid, in the solid and over the whole domain (Supplementary Table S2). The interface kink therefore leaves the Taylor–Hood temperature order intact. The discrete interface flux jump $[\,k\,\partial_n T\,]_\Gamma$ converges at $\approx \mathcal{O}(h^2)$, one order lower, as expected for the P_2 gradient.

Supplementary Table S2. Conjugate-heat-transfer MMS. GNN P_2 energy operator, $k_s/k_f = 0.1$. $L^2(T)$ is over the whole domain, the fluid ($|_f$) and the solid ($|_s$); $[k\partial_n T]_\Gamma$ is the interface flux jump and "ord" the order between adjacent rows. The observed orders, 3.58–3.74, exceed the theoretical 3. The test therefore bounds any interface-induced loss of order only to within that excess.

	$h_{\min}$	$L^2(T)$	ord	$L^2(T)\|_f$	$L^2(T)\|_s$	$[k\partial_n T]_\Gamma$
2D	3.13×10^{-2}	1.18×10^{-7}	–	8.14×10^{-8}	8.58×10^{-8}	9.79×10^{-5}
	1.56×10^{-2}	9.61×10^{-9}	3.62	6.69×10^{-9}	6.90×10^{-9}	2.48×10^{-5}
	7.81×10^{-3}	8.05×10^{-10}	3.58	5.64×10^{-10}	5.74×10^{-10}	6.23×10^{-6}
3D	1.25×10^{-1}	4.65×10^{-5}	–	3.29×10^{-5}	3.29×10^{-5}	1.14×10^{-3}
	8.33×10^{-2}	1.03×10^{-5}	3.72	7.23×10^{-6}	7.34×10^{-6}	4.24×10^{-4}
	6.25×10^{-2}	3.51×10^{-6}	3.74	2.45×10^{-6}	2.52×10^{-6}	2.16×10^{-4}

High conductivity contrast. Temperature still converges at $\gtrsim\mathcal{O}(h^3)$ when the solid is the better conductor, as in the engine example (Supplementary Note S6). At $k_s/k_f = 10^2$ and 10^3 the orders are $\approx$4.0 in 2D and $\approx$3.9 in 3D, and the flux jump converges at $\mathcal{O}(h^2)$ (Supplementary Table S3). The flux-jump magnitude scales with k_s, and its order does not change.

Supplementary Table S3. Conjugate-heat-transfer MMS at high conductivity contrast. As Supplementary Table S2, with the solid more conductive, $k_s/k_f = 100$ and 1000. Errors are at the finest mesh; ord(T) and ord(jmp) are the orders of $L^2(T)$ and of $[k\partial_n T]_\Gamma$ between the two finest meshes.

	k_s/k_f	$h_{\min}$	$L^2(T)$	ord(T)	$[k\partial_n T]_\Gamma$	ord(jmp)
2D	100	7.81×10^{-3}	1.05×10^{-8}	4.00	1.35×10^{-3}	2.02
	1000	7.81×10^{-3}	1.06×10^{-7}	3.99	1.35×10^{-2}	2.02
3D	100	6.25×10^{-2}	1.40×10^{-4}	3.88	1.48×10^{-1}	2.21
	1000	6.25×10^{-2}	1.40×10^{-3}	3.88	1.48	2.21

Two-material conduction slab. The conjugate assembly used by the engine example reproduces the analytic interface temperature $q^*/[2(k_s + k_f)]$ of a two-material slab with a prescribed nondimensional interface flux q^*. The relative error is $< 10^{-13}$ for $k_s/k_f \in \{1, 10^2, 2\times10^3\}$, because the P_2 space contains the piecewise-linear exact field.

S1.4 Graph execution of the multigrid cycle

The V-cycle is symmetric positive definite. The V-cycle preconditioner is symmetric positive definite, as conjugate gradients requires. Restriction is the exact transpose of prolongation at every level, the pre- and post-smoothing counts are equal, and the smoother is symmetric. With B one V-cycle, $\langle Bx, y\rangle = \langle x, By\rangle$ holds to 3×10^{-16}

(weighted Jacobi) and 1×10^{-16} (Chebyshev) for random x, y on the production operators. On the test operators shipped with the code package the weighted-Jacobi cycle is symmetric to 2×10^{-17}, and the Rayleigh quotient $\langle x, Bx\rangle/\langle x, x\rangle$ is positive for every random vector tested, with minimum 0.03 (code package, experiment `vcycle_spd`).

The graph execution reproduces the cycle it represents. The graph execution reproduces a reference V-cycle to a relative L^2 difference of 1.5×10^{-16} to 3.8×10^{-16}, the fp64 round-off of the several dozen sparse applications in one cycle. The reference is written in `scipy` and runs on the hierarchy extracted from the graph object's own layers. It uses the same weighted-Jacobi smoother, smoothing counts and coarsest-level relaxation. The agreement holds for $\omega \in \{0.5, 0.7, 0.9\}$ and $(n_{\text{pre}}, n_{\text{post}}) \in \{(2,2), (1,1), (3,1)\}$ on 32^2, 64^2 and 128^2 Poisson operators with four and five levels. PyAMG's default cycle cannot serve as the reference, because it smooths with Gauss–Seidel and solves the coarsest level directly. On the test operators shipped with the code package the same comparison gives 1.7×10^{-17} to 2.1×10^{-16} (code package, experiment `vcycle_identity`).

Rebuilding the hierarchy. The identity is measured against the hierarchy stored in the graph object. A fresh smoothed-aggregation build from the same operator differs in its coarse operators by 1.0×10^{-4} to 1.6×10^{-2} in relative max norm unless the random seed of its spectral-radius estimate is fixed. With a fixed seed, two builds on the package operators are identical to the last bit. Different seeds leave the iteration counts unchanged and move the converged solution by less than 10^{-9} in relative norm (code package, experiment `hierarchy_rebuild`).

S1.5 Setup

The 2D GNN meshes are Gmsh v4.13 triangulations [1]; the 3D meshes are tetrahedralizations of a uniform cube. The CNN uses the production Cartesian grid. SIMPLE uses the production under-relaxation $\alpha_U/\alpha_p/\alpha_T = 0.7/0.3/0.9$ and relative tolerance 10^{-6}. The MMS runs use the production pseudo-time step $10^5\, h^2/\mathrm{Pr}$ (Supplementary Table S23). At the outer-iteration cap of 5,000 to 20,000 the residual sits at its floor of 10^{-5} to 10^{-10}, and orders are read above that floor. Tightening the energy tolerance changes the errors in the fifth figure at most.

S2 Differentiating the solver

Both differentiation routes give verified gradients of the coupled solver. The unrolled gradient matches central finite differences to 2.6×10^{-9} to 2.7×10^{-6} but approaches the

converged-solution sensitivity slowly. The implicit gradient removes most of that error and agrees with pyadjoint to 4×10^{-7} to 2×10^{-5}.

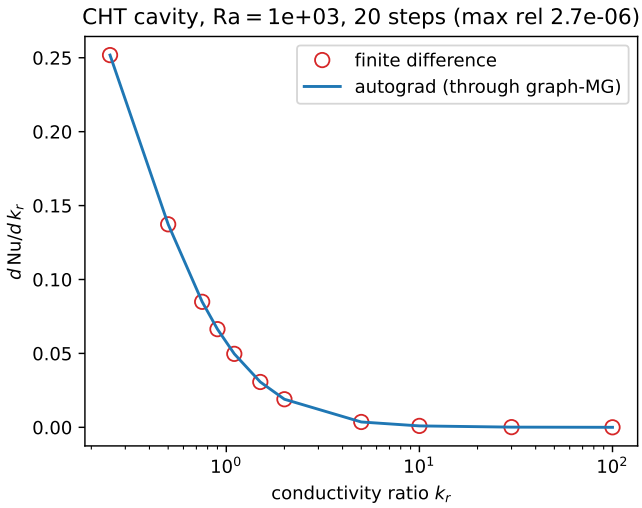
Unrolled reverse mode differentiates the executed fixed-iteration program and stores every linear-solver iterate on the tape. *Implicit* differentiation differentiates the residual of each linear solve at its solution with one transposed solve, and so targets the sensitivity of the converged discrete solution. Four limits apply throughout. The multigrid hierarchy is built once, outside the differentiated graph. The unrolled route differentiates its application; the implicit route uses a residual-based backward rule and a transposed solve without differentiating the Krylov iteration history. The runs are single-GPU, because the in-place halo exchange breaks the tape. The verification and cost runs use fixed linear-solve iteration counts rather than the production stopping tolerance; the pyadjoint comparison stops both codes at 10^{-8}. Transfer to the tolerance-controlled production configuration is not established. The optional Picard momentum path, a nonlinear fixed point whose adjoint is that of the nonlinear system, is not addressed.

S2.1 Verification of the differentiated program

Unrolled reverse mode gives a sensitivity $d\mathrm{Nu}/dk_r$ that agrees with a central finite difference to a relative 2.6×10^{-9} to 2.7×10^{-6} across $k_r \in [0.25, 100]$ (Supplementary Figure S2). The largest errors are the finite-difference floor, where the sensitivity is $\mathcal{O}(10^{-5})$. The case is the full coupled rotational-IPCS + RK3–CN solve of the conjugate cavity over 20 steps, with hot-wall Nusselt number $\mathrm{Nu} = g^\top T$. The thermal stiffness is exactly affine in the solid–fluid conductivity ratio, $K_T(k_r) = K_{\mathrm{fluid}} + k_r\, K_{\mathrm{band}}$, with K_{band} the stiffness of the solid band. The unrolled tape spans the whole coupled step, including the graph-multigrid pressure V-cycle, which the gradient reaches through its buoyancy-driven right-hand side. Every linear solve uses a fixed iteration count, so both computations evaluate the same smooth discrete map. The agreement verifies the derivative of that fixed-iteration program, which can differ from the sensitivity of the converged discrete solution.

S2.2 Convergence of the unrolled derivative to the discrete sensitivity

Differentiating through n fixed Krylov sweeps converges slowly, and not monotonically, to the converged-solution derivative. A direct adjoint, free of iteration error, gives that derivative and agrees with a central difference of the converged solve to 2.3×10^{-8} at $k_r = 2$ on the 513-node mesh. It is taken on one implicit Crank–Nicolson conjugate step: with $\mathsf{A}(k_r)x = b(k_r)$ and $\mathrm{Nu} = g^\top x$, solving $\mathsf{A}^\top \lambda = g$ gives $d\mathrm{Nu}/dk_r = \lambda^\top(\partial_{k_r} b - \partial_{k_r}\mathsf{A}\, x^*)$. Against this reference, at $k_r = 2$ the unrolled error is 114% at $n = 20$ and first reaches



Supplementary Figure S2. Unrolled reverse mode against central finite differences. Sensitivity $d\mathrm{Nu}/dk_r$ of the hot-wall Nusselt number, conjugate cavity at $\mathrm{Ra} = 10^3$, by unrolled reverse mode through the full coupled solve (line) and central difference (markers).

10^{-6} at $n = 80$. At $k_r = 100$ the unrolled derivative is still off by a factor of 13.5 at $n = 80$ and converges only by $n = 200$. On the 513-node mesh at $k_r = 2$ a central difference through the same program agrees with the unrolled derivative to between 10^{-5} and 10^{-8}, so it cannot reveal this error.

S2.3 Implicit differentiation

Single thermal step. On the same step the implicit rule lowers the error at $n = 20$ from 114% to 9.1×10^{-7}, so 10^{-6} is reached at $n = 20$ rather than $n = 80$. Its floor is 6.5×10^{-15} against the unrolled 5.3×10^{-11}. Writing the solve as $r(x, \theta) = \mathsf{A}(\theta)x - b(\theta) = 0$ and differentiating the residual at fixed x gives, for a scalar output L, $dL/d\theta = -\lambda^\top \partial_\theta r$ with $(\partial_x r)^\top \lambda = \partial_x L$. The backward pass stores one residual and takes one transposed solve, and its error falls monotonically with the forward residual. The transposed solve must converge in its own right: on the 513-node mesh it needs about 60 iterations to bring the implicit gradient to a relative error of 10^{-8} (8×10^{-4} at 40 iterations).

Coupled step. At converged counts and $k_r = 2$ the implicit and unrolled gradients agree to 2×10^{-12} over a twenty-step window and 1×10^{-12} over forty. Each is within 1.2×10^{-11} of a central finite difference at twenty steps and 2.9×10^{-9} at forty. The rule covers all six linear solves per Runge–Kutta substage; only the thermal solve carries k_r. Nu agrees between the routes to about 10^{-14}, the solver's reproducibility floor.

At the prescribed forward count of $n = 40$ iterations per solve, the solves are not converged; the thermal residual is 3.6×10^{-6}. Against the converged sensitivity, taken with converged subsolves over the same window, the implicit gradient is 5.1×10^{-4} off

at $k_r = 100$ and the unrolled one 25%. At $k_r = 2$ the two are 4.5×10^{-8} and 3.9×10^{-6} off. The implicit error remains because the backward is evaluated at the inexact forward state x_n, which enters the $\partial_{k_r}\mathsf{A}$ term. What the rule removes is differentiation through the truncated iteration. That error is the larger one here, by two to three orders of magnitude. The transposed solves use four times the forward iteration count, which the costs in Supplementary Note S2.4 include.

S2.4 Cost and memory

Single thermal step, unrolled against implicit. From 8.4×10^3 to 1.2×10^6 DOF the implicit gradient costs 2.0 to 2.9 forward solves, a weak dependence on problem size. The unrolled gradient rises from 7.4 to 29.3, because the cost of each of its fixed iterations grows with the mesh, not because the iteration count grows. Peak memory at the largest size is 10.7 GiB implicit and 35.5 GiB unrolled. These numbers are not comparable with the coupled costs or with Supplementary Table S4.

Coupled step, unrolled against implicit. On the m1 mesh, over the coupled window of Supplementary Note S2.3 and timed at $k_r = 100$, the implicit sweep costs 4.8 to 4.9 forward solves and the unrolled one 5.4 to 5.7. Implicit peak memory stays at 2.720 GiB from 40 to 320 iterations, while the unrolled tape grows 7.56 $\rightarrow$ 22.14 $\rightarrow$ 41.57 GiB. At 2000 iterations the unrolled route exhausts the 141 GB H200; the implicit one is unchanged.

Coupled step, several conductivities. Each reverse-mode gradient matches central differences componentwise to 5.1×10^{-9} to 8.1×10^{-8} before its cost is recorded, in the same run as the plotted costs. The solid band is cut into M layers with their own conductivities, so $K_T(\boldsymbol{k}) = K_{\text{fluid}} + \sum_{j=1}^{M} k_j K_{\text{seg},j}$ stays affine. At $M = 1$ it reproduces the scalar operator to 4.1×10^{-15} in Nu and 7.3×10^{-15} in the gradient. Main-text Fig. 2c gives the resulting costs against M.

S2.5 Comparison with pyadjoint on DOLFINx

Our unrolled gradient is delivered 2.2× faster than that of pyadjoint on DOLFINx [2, 3] at the coarsest mesh and 5.9× at the finest (Supplementary Table S4; versions in Supplementary Table S24). Both codes solve the same discrete conjugate thermal step, with the same mesh, forms, Crank–Nicolson window, hot-wall functional and linear-solve tolerance. The tolerance is matched; the gradient accuracy is not. Their Nusselt numbers agree to 8×10^{-6}–1×10^{-4}. The implicit route serves only as an accuracy reference: its gradient agrees with pyadjoint's to 4×10^{-7}–2×10^{-5}. The two are independent rules for

the same converged sensitivity. The unrolled gradient sits about 10^{-3} from both, which is the truncation it differentiates.

Unrolling costs 5.2–6.9 forward solves because it differentiates through the linear-solver iterations, whereas pyadjoint solves one adjoint linear system and costs a near-constant 1.7. Our forward solve is 7 to 23× faster, which more than covers the overhead.

Unrolled memory grows about 2.6× per doubling of DOF, which stops the comparison at 9.4×10^4 DOF on the 141 GB H200. The tape holds every solver iterate: 3.7 GiB at 1.2×10^4 DOF, 65.2 GiB at 9.4×10^4. The next mesh would need roughly 170 GiB. In this comparison pyadjoint avoids the iteration-history storage that unrolling requires. The implicit route, which stores one residual per solve (Supplementary Note S2.3), also avoids that storage, but its runtime is not compared with pyadjoint here.

Supplementary Table S4. Unrolled reverse-mode automatic differentiation (AD) against pyadjoint on DOLFINx. Conjugate thermal step, 20 Crank–Nicolson steps, one solid-layer conductivity ($M = 1$), forward and adjoint linear-solve tolerance 10^{-8}, median of three; ours on one H200, pyadjoint on 32 CPU cores. DOF is the P_2 temperature count. Doubling the pyadjoint allocation from 16 to 32 cores changed its time by −3 to +6%; repeating each row at $M = 8$ moved no column by more than 4.9%. "ovh" is gradient over forward time; "ratio" is pyadjoint gradient time over ours.

	ours (GPU, unrolled)				pyadjoint (CPU)		
DOF	fwd (s)	grad (s)	ovh	tape (GiB)	grad (s)	ovh	ratio
11963	0.24	1.23	5.23	3.7	2.75	1.64	2.24
23899	0.33	1.72	5.26	9.7	5.13	1.62	2.98
46929	0.45	2.54	5.60	26.0	10.09	1.73	3.97
94291	0.57	3.99	6.93	65.2	23.50	1.79	5.90

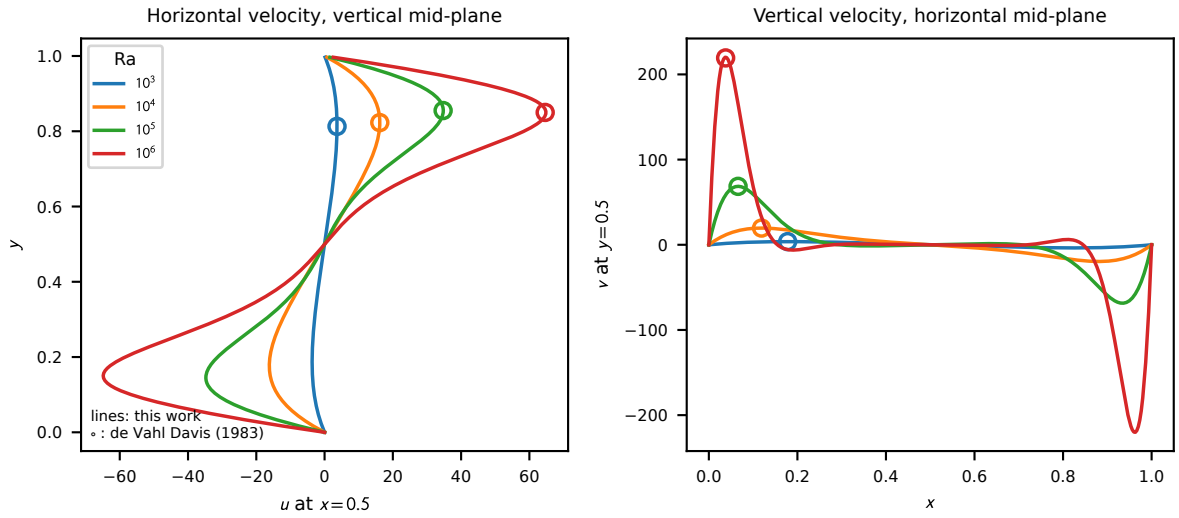
S3 Cavity and Rayleigh–Bénard benchmark record

In 2D the GNN matches the benchmark Nusselt numbers to within 0.3%, and the spectral value at $\mathrm{Ra} = 10^6$ to within 0.006% (Supplementary Table S7). Its centreline velocity extrema agree to within 1.5% (Supplementary Figure S3). The 2D Nusselt table is Extended Data Table 1. In 3D, Supplementary Table S5 compares the converged Nusselt numbers of all four solvers with the pseudo-spectral reference of Tric et al. [4], and Supplementary Table S6 gives the measured 2D wall-clock times to convergence and the reconstructed 3D times. The reference Nusselt values are from de Vahl Davis [5], Le Quéré [6], Tric [4] and Johnston–Doering [7].

Stability at $\mathrm{Ra} = 10^8$**.** On the 2D Rayleigh–Bénard cell at h_{128}, our explicit CNN configuration diverges at $t \approx 0.040$, before the flow develops. Its step is held to the advective limit, $\Delta t \approx 9.3 \times 10^{-8}$. From the same initial condition the semi-implicit RK3–CN scheme on the GNN stays stable and reaches $\langle \mathrm{Nu} \rangle = 25.6$ to 25.7 (three seeds) by

Supplementary Table S5. Converged Nu on the 3D Boussinesq cavity benchmark. The reference column is the pseudo-spectral average Nu of Tric et al. [4].

Ra	mesh	CNN	GNN	FEniCSx	OpenFOAM	Tric (2000)
10^3	h_{16}	1.078	1.072	1.072	1.076	1.0700
	h_{32}	1.073	1.071	1.071	1.072	
	h_{64}	1.071	1.071	1.071	1.071	
10^4	h_{16}	2.152	2.072	2.072	2.129	2.0542
	h_{32}	2.077	2.058	2.058	2.075	
	h_{64}	2.060	2.053	2.056	2.060	
10^5	h_{16}	5.478	4.504	4.504	4.826	4.3371
	h_{32}	4.512	4.375	4.375	4.480	
	h_{64}	4.377	4.342	4.343	4.373	



Supplementary Figure S3. Centreline velocity profiles against the de Vahl Davis (1983) benchmark. Horizontal velocity u on the vertical mid-plane $x = 0.5$ (left) and vertical velocity v on the horizontal mid-plane $y = 0.5$ (right), in the κ/L velocity scaling of the benchmark. Solid lines: GNN–SIMPLE at h_{128}; open circles: the reference extrema.

$t = 0.3$ to 0.5. This shows the stability margin of implicit diffusion for this configuration only. It does not show that explicit structured methods fail at $\mathrm{Ra} = 10^8$ in general.

S3.1 Degrees of freedom of the cavity meshes

At a fixed mesh the P_2–P_1 solvers (GNN, FEniCSx) carry $\approx 3.8\times$ more unknowns than the colocated solvers (CNN, OpenFOAM) in 2D. In 3D they carry 5.0 to 5.9× more. Totals over all fields $u, v, (w,)\, p, T$ do not depend on Ra. In 2D, colocated against P_2–P_1, they are 16384 against 62780 at h_{64}, 65536 against 248575 at h_{128} and 262144 against 989664 at h_{256}. In 3D they are 20480 against 121576 at h_{16}, 163840 against 873114 at h_{32} and 1310720 against 6614191 at h_{64}. Extended Data Fig. 4b plots the per-step cost of the structured and unstructured solvers against total DOF.

Supplementary Table S6. Wall-clock time to convergence (minutes) at matched mesh labels. Tier C (main-text Section 4.2): the PDE and the outer controls are shared, and each column times a complete solver stack. CNN and GNN on one H200, FEniCSx and OpenFOAM v12 on 32 CPU cores. The 2D block is measured; the 3D block is reconstructed as iterations to convergence times the isolated per-iteration cost, and the two directly timed FEniCSx h_{64}, Ra $\geq 10^4$ cells agree with it to 0.1%. For the GNN the directly measured walls of the same 3D runs are 1.1 to 1.9 times the reconstructed values, because the developed-flow cost per iteration exceeds the isolated cost; the reconstruction is kept so that all four solvers share one basis. FEniCSx at h_{64} departs from the shared controls twice: Ra $= 10^4$ uses $\alpha_U = 0.5$. The ‡ entry (Ra $= 10^3$) is Anderson-accelerated, so it is a lower bound on the plain-SIMPLE time.

Ra	mesh	CNN	GNN	FEniCSx	OpenFOAM
Two-dimensional benchmark (directly measured)					
	h_{64}	9.1	1.5	16.9	0.5
10^4	h_{128}	25.2	3.3	123.4	3.9
	h_{256}	70.5	42.6	838.4	57.2
	h_{64}	10.3	0.7	16.4	0.6
10^5	h_{128}	34.3	3.9	135.3	3.6
	h_{256}	99.2	24.4	854.5	51.9
	h_{64}	18.9	0.9	19.2	0.6
10^6	h_{128}	38.5	3.1	142.4	3.3
	h_{256}	131.9	12.9	876.6	44.8
Three-dimensional benchmark (reconstructed)					
	h_{16}	6.1	0.5	6.7	0.2
10^3	h_{32}	16.9	2.1	130.7	1.8
	h_{64}	55.2	48.6	808.2 ‡	45.6
	h_{16}	4.9	0.2	6.0	0.2
10^4	h_{32}	12.4	1.6	112.3	2.3
	h_{64}	41.0	11.9	3680.5	40.6
	h_{16}	7.2	0.3	7.9	0.3
10^5	h_{32}	13.1	1.0	114.4	2.2
	h_{64}	36.7	23.0	2350.4	35.8

Supplementary Table S7. Validation of the 2D converged Nusselt number against established benchmarks. Differentially heated square cavity, air (Pr = 0.71). "This work" is the GNN–SIMPLE value at h_{256}. The references are the mean wall Nusselt number of de Vahl Davis [5] and, at Ra $= 10^6$, the spectral solution of Le Quéré [6].

Ra	This work (GNN)	de Vahl Davis (1983)	Le Quéré (1991)
10^3	1.117	1.118	–
10^4	2.245	2.243	–
10^5	4.519	4.519	–
10^6	8.8257	8.800	8.8252

S3.2 Statistical convergence of the turbulent Rayleigh–Bénard Nusselt number

The time-averaged $\langle \mathrm{Nu} \rangle$ of the turbulent Rayleigh–Bénard cell in main-text Section 2.4 ($\Gamma = 2$, Ra $= 10^7$, Pr $= 0.71$) is converged in time by the post-run blocking analysis

below. The driver's own windowed stationarity test had not fired at 1.5 diffusion times. One GNN trajectory at h_{128} spans 1.5 thermal-diffusion times, about 4.0×10^3 free-fall turnovers, with the plate Nusselt numbers logged every ten steps.

Time average and confidence interval. After a 0.15-diffusion-time warm-up, the remaining $\approx 3.6 \times 10^3$ turnovers give

$$\langle \mathrm{Nu} \rangle = 14.04 \pm 0.025 \quad (95\% \text{ confidence}), \tag{S4}$$

2.9% above the 2D direct-numerical-simulation correlation of Johnston and Doering, $0.138\,\mathrm{Ra}^{0.285} = 13.64$ [7]. The Nusselt number decorrelates within $\tau_{\mathrm{dec}} \approx 0.9$ turnovers, and its autocorrelation carries a large-scale-circulation oscillation of period ≈ 4.5 turnovers. That oscillation makes the integrated autocorrelation time unreliable, so the interval comes from Flyvbjerg–Petersen blocking [8]. It is the largest interval over block lengths leaving at least 50 blocks, reached at ≈ 1.1 turnovers, and is therefore conservative. The offset is more than ten times the half-width of the interval, so it is not a temporal-sampling artefact. The offset is comparable to the few-percent inter-study scatter of 2D $\Gamma = 2$ Rayleigh–Bénard Nu [9].

Insensitivity to warm-up and window. Varying the warm-up from 0.05 to 0.30 diffusion times (133 to 799 turnovers) moves $\langle \mathrm{Nu} \rangle$ by less than 0.002. Lengthening the window from 1199 to 3597 turnovers narrows the interval from ± 0.043 to ± 0.025, with the mean within 0.007.

Grid convergence and cross-code agreement. Under refinement the two discretizations approach each other from opposite sides, and at their finest meshes both lie near 14.0, within each other's seed intervals (Supplementary Table S8). At those meshes they lie 2.7 to 3.8% above the Johnston–Doering correlation. These runs use a short protocol, a 0.05-diffusion-time warm-up and a 0.25-diffusion-time window. This protocol returns the same mean as the long window for a given trajectory. Realizations still differ by about 1% because the large-scale circulation settles into long-lived states, so each entry is a mean over independent seeds. The long GNN trajectory above, 14.04, lies 1.6 seed standard deviations below the six-seed h_{128} mean 14.15 ± 0.07, so seed scatter dominates the uncertainty. Neither sequence is Richardson-extrapolated, and the CNN is still rising at h_{512}.

Wall-clock time to each solver's own converged statistic. On one H200 the CNN reaches its stationary $\langle \mathrm{Nu} \rangle$ in 1.15 h at h_{64} and 2.13 h at h_{128}; the GNN in 1.4 h and 5.5 h. Each wall-clock time is reconstructed as the three-seed mean step count of the production runs (64.9k and 116.8k steps) times the cost per step of Supplementary

Supplementary Table S8. Mesh dependence of the turbulent Rayleigh–Bénard $\langle \mathrm{Nu} \rangle$ ($\Gamma = 2$, $\mathrm{Ra} = 10^7$, $\mathrm{Pr} = 0.71$, short protocol). Seed-ensemble mean and 95% interval across the n seeds of each entry. $\Delta\%$ is relative to the Johnston–Doering correlation, 13.64. The GNN was not run at h_{512}.

	CNN (structured FD)			GNN (P_2–P_1)		
Mesh	$\langle \mathrm{Nu} \rangle$	$\Delta\%$	n	$\langle \mathrm{Nu} \rangle$	$\Delta\%$	n
h_{64}	13.22 ± 0.09	-3.1	6	14.65 ± 0.02	$+7.4$	6
h_{128}	13.52 ± 0.01	-0.9	6	14.15 ± 0.07	$+3.7$	6
h_{256}	13.80 ± 0.24	$+1.2$	6	14.16 ± 0.18	$+3.8$	4
h_{512}	14.01 ± 0.20	$+2.7$	6			

Table S12. A direct three-seed run of the GNN at h_{64} with the same settings took 1.44 to 1.46 h including set-up. The two solvers reach different stationary values at a given mesh (Supplementary Table S8), so these are costs to each solver's own statistic, not to a shared target. At matched resolution the colocated CNN carries $\approx 3.3\times$ fewer unknowns than the P_2–P_1 GNN.

S4 Conjugate cavity validation

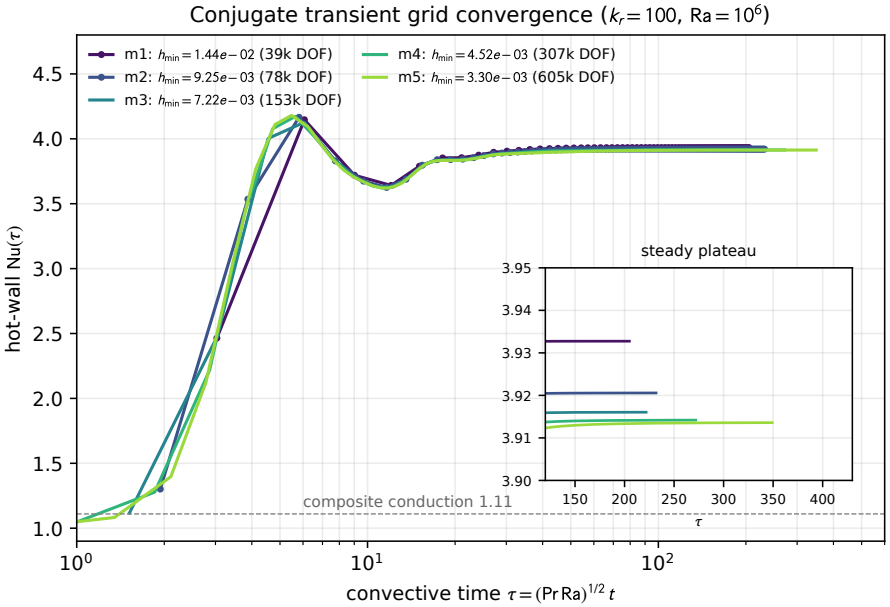
The steady hot-wall Nusselt number matches the centred-conducting-body benchmark of House et al. [10], which has a two-dimensional fluid–solid interface, to within 0.23% in four of five cases and 1.6% in the fifth. This note also gives the grid convergence behind the conjugate rows of main-text Table 1; the main-text validation against Khatamifar et al. [11] is in main-text Section 2.5.

Centred conducting body. Supplementary Table S9 lists the five cases. The 1.6% case is the large, strongly conducting body ($\zeta = 0.9$, $k_r = 5$). The solid is a centred square of side ζ, so the flow must pass around a two-dimensional interface. At $\zeta = 0.9$ the fluid gap is 0.05 wide on each side. The streamfunction extremum agrees to within $\sim 1\%$ where tabulated. The case is steady and source-free, so the two wall Nusselt numbers must be equal, and $|\mathrm{Nu}_h - \mathrm{Nu}_c|/\mathrm{Nu} < 10^{-4}$ in every case, at the sample the values are read from. The $\zeta = 0.9$ cases are grid-converged; the insulating one ($k_r = 0.2$) changes by 0.12% from h_{160} to h_{240}. In every case the temperature is a steady conjugate energy solve on the flow at the end of a fixed run (60000 steps at $\zeta = 0.5$, 80000 at $\zeta = 0.9$), because the solid equilibrates far more slowly than the flow. These runs use a pressure tolerance of 10^{-7} with cap 80 and a mass tolerance of 10^{-8}.

Transient grid convergence. After the start-up overshoot of Nu, for convective time $\tau = (\mathrm{Pr}\,\mathrm{Ra})^{1/2} t \geq 10$, the hot-wall Nu histories on the three finest meshes agree to

Supplementary Table S9. Validation against the centred-conducting-body benchmark of House et al. [10]. Steady hot-wall Nu with a centred square solid of side ζ and conductivity ratio $k_r = k_s/k_f$, Pr = 0.71; the $\zeta = 0.9$ values use the h_{240} mesh. Hot- and cold-wall values agree to four significant figures, so one Nu is given. The differences use the unrounded reference values (4.6257, 4.5061, 4.3221, 2.402 and 3.868).

ζ	Ra	k_r	This work (GNN)	House (1990)	diff.
0.5	10^5	0.2	4.626	4.626	+0.02%
0.5	10^5	1.0	4.506	4.506	+0.00%
0.5	10^5	5.0	4.319	4.322	−0.08%
0.9	10^6	0.2	2.408	2.402	+0.23%
0.9	10^6	5.0	3.806	3.868	−1.61%



Supplementary Figure S4. Grid convergence of the transient conjugate development. Hot-wall Nu against convective time $\tau = (\mathrm{Pr}\,\mathrm{Ra})^{1/2}t$, with t the diffusive time, for $k_r = 100$, Ra = 10^6 and partition thickness $T_p = 0.1$ on meshes m1 to m5; the inset shows the steady plateau. The dashed line is the composite-conduction equilibrium $\mathrm{Nu}_{\mathrm{cond}} = 1/[(1 - T_p) + T_p/k_r] = 1.11$; runs start from the linear profile at Nu = 1, and the $t = 0$ sample is off the logarithmic axis. Sparse coarse-mesh samples are drawn as markers.

within 0.3% at fixed τ (Supplementary Figure S4). The case is $k_r = 100$ at Ra = 10^6, the strongest convection in the validation set, on the five-mesh $\sqrt{2}$ sequence m1 to m5. During the rise the samples differ by up to 6%. Part of this is sampling: Nu is logged every 200 steps, which is 3.0, 1.5 and 0.7 convective units on m1, m3 and m5. The steady plateau falls monotonically from Nu = 3.933 on m1 ($h_{\min} = 1.4 \times 10^{-2}$) to 3.913 on m5 ($h_{\min} = 3.3 \times 10^{-3}$). Richardson extrapolation of m3 to m5 gives 3.913, the value in Extended Data Table 2, and the m3 to m5 plateaus span 0.06%.

Grid-convergence inputs behind main-text Table 1. On m5 both time integrators, RK3–CN and the implicit-explicit scheme ARS(2,2,2), give hot-wall Nusselt num-

bers that agree to four significant figures with the grid-converged 1.059, 1.751 and 2.794 of Extended Data Table 2. Supplementary Table S10 lists the per-mesh values, and main-text Table 1 quotes the RK3–CN entries. They are re-evaluated from the saved double-precision fields, because the four-decimal solver log is too coarse to difference. The RK3–CN columns at $\mathrm{Ra} = 10^4$ and 10^5 were read at the production stop time, $t = 0.4$ to 0.9, and the ARS(2,2,2) columns at a settled state. At $\mathrm{Ra} = 10^5$ the successive differences are of the order of the steady-state tolerance. At $\mathrm{Ra} = 10^4$ they do not shrink, so no order is fit and main-text Table 1 quotes the five-mesh spread. Read at a settled state the two schemes agree to 8×10^{-6} at $\mathrm{Ra} = 10^4$ and 6×10^{-9} at $\mathrm{Ra} = 10^5$, and the $\mathrm{Ra} = 10^5$ ladder gives order 3.0 and GCI 0.002%.

Supplementary Table S10. Per-mesh conjugate Nusselt numbers behind main-text Table 1. Hot-wall Nu ($k_r = 1$, $T_p = 0.1$) on meshes m1 to m5 (characteristic length l_c in parentheses) for both integrators, from the saved double-precision fields. Observed order p, Nu_∞ and the grid convergence index (GCI; safety factor 1.25, $r = \sqrt{2}$) use m3 to m5, with GCI also given at assumed $p = 2$ and 3. A dash marks an order that could not be fit from these values. The last row is the largest relative difference between the integrators over m1–m5. The RK3–CN columns at $\mathrm{Ra} = 10^4$ and 10^5 are read at the production stop time ($t = 0.4$ to 0.9) and the ARS(2,2,2) columns at a settled state. The RK3–CN ladders at $\mathrm{Ra} = 10^4$ and 10^5 and all ARS ladders solved the pressure with five fixed weighted-Jacobi V-cycles per solve, and 11 of the 15 ARS runs ran to a fixed step count; the published ARS $\mathrm{Ra} = 10^6$ m4 and m5 runs stopped at $t = 0.18$, before Nu settled.

Mesh (l_c)	$\mathrm{Ra} = 10^4$	$\mathrm{Ra} = 10^5$	$\mathrm{Ra} = 10^6$
RK3–CN			
m1 (0.020)	1.059 482 8	1.752 432 1	2.807 482 6
m2 (0.0141)	1.059 421 1	1.751 346 8	2.799 184 1
m3 (0.010)	1.059 375 7	1.750 969 6	2.796 042 1
m4 (0.00707)	1.059 274 1	1.750 837 5	2.794 810 4
m5 (0.005)	1.059 148 4	1.750 800 7	2.794 362 1
observed order p (m3–m5)	–	3.68	2.92
Nu_∞ (Richardson)	–	1.750 786 5	2.794 105 5
GCI at m5, observed p (%)	–	0.0010	0.0115
GCI at m5, assumed $p = 2$ (%)	0.0148	0.0026	0.0201
GCI at m5, assumed $p = 3$ (%)	0.0081	0.0014	0.0110
spread m1–m5 (%)	0.032	0.093	0.470
ARS(2,2,2)			
m1 (0.020)	1.059 488 6	1.752 431 8	2.807 482 6
m2 (0.0141)	1.059 447 2	1.751 343 2	2.799 184 0
m3 (0.010)	1.059 433 2	1.750 960 2	2.796 042 1
m4 (0.00707)	1.059 427 7	1.750 815 0	2.794 820 4
m5 (0.005)	1.059 416 7	1.750 763 9	2.794 370 6
observed order p (m3–m5)	–	3.01	2.88
Nu_∞ (Richardson)	–	1.750 736 1	2.794 108 5
GCI at m5, observed p (%)	–	0.0020	0.0117
GCI at m5, assumed $p = 2$ (%)	0.0013	0.0037	0.0201
GCI at m5, assumed $p = 3$ (%)	0.0007	0.0020	0.0110
spread m1–m5 (%)	0.007	0.095	0.469
scheme difference, max over m1–m5	2.5×10^{-4}	2.1×10^{-5}	3.6×10^{-6}

S5 Same-discretization throughput and the pressure solve

On each of the four matched problems the CPU/GPU per-step ratio grows with mesh size, from 0.81 on the coarsest engine mesh to 181.2 on the finest heat-sink mesh (Supplementary Tables S11 to S14). They are Tier B, as defined in main-text Section 4.2. Graph multigrid on one H200 GPU is compared with FEniCSx with PETSc CG + GAMG on 32 CPU cores. Discretization, mesh, time step and tolerances are shared (10^{-6} for pressure, 10^{-10} for mass and diffusion); only the linear-algebra backend and the device change. On the heat-sink m3 mesh about one GPU mass solve in ten stops at the driver's 300-iteration cap near 10^{-9}, which changes the step time by under 1%. Mass and diffusion dominate the CPU step on the engine and heat sink. Had the GPU mass and diffusion tolerance been left at 10^{-7} instead of the shared 10^{-10}, the ratio would be 1.25 to 1.26 times larger on the engine and 1.29 to 1.34 times larger on the heat sink. The conjugate-cavity GPU runs use CUDA-graph capture, worth about 2.2× on the 2D cavity at h_{128} (code package, table `mgsolver`). The engine and heat-sink drivers run without capture, and its effect there was not measured. Engine and heat-sink throughput timings are medians of three repetitions on an exclusive node.

Conjugate cavity. Across the six conjugate meshes the GPU per-step cost over steps 200 to 600 from rest rises from 40.5 to 60.75 ms (Supplementary Table S11). It falls once the flow settles, because the mass and diffusion solves need fewer iterations. At the finest mesh the GPU reaches a steady Nu in 108.2 min, against a projected 2588.0 min on the CPU. The CPU run is assumed to take the same step count, set by the shared scheme and time step. The GPU run time is measured; the CPU run time is projected as the GPU run time times the per-step ratio of these probes. This assumes that ratio holds over the whole trajectory, which was not measured.

Engine cylinder. On the body-fitted engine the CPU/GPU ratio rises from 0.81 on the coarsest mesh to 23.9 on the finest (Supplementary Table S13). The GPU per-step cost rises 1.5× across a 21× size range, while the CPU sparse solve grows with the problem. FEniCSx is the only comparable reference, since it alone shares the P_2–P_1 discretization on this curved mesh. The structured CNN would have to staircase the interface, which would confound geometry with discretization. Both implementations solve the same problem: at the same step count the peak interface temperature θ agrees to all four printed digits. The θ columns of Supplementary Table S13 are read at the end of each run, step 200 on the GPU and step 60 on the CPU, and differ by at most 0.22%.

Supplementary Table S11. Same-scheme conjugate throughput: per step and to a converged Nu ($k_r = 1$, $T_p = 0.1$, Ra $= 10^6$). Per-step costs are measured in each implementation's cold-start probe, as the wall-clock slope over steps 200 to 600 on the GPU and 60 to 200 on the CPU, both legs in one job per mesh on an exclusive node. Steps is the count to the shared criterion $|\Delta\mathrm{Nu}/\mathrm{Nu}| < 10^{-5}$. The speedup is the per-step ratio and, by construction, the run-time ratio.

	per step, measured (ms)			to converged Nu (min)		
Mesh (total DOF)	GPU	CPU	Steps	GPU, measured	CPU, projected	Speedup
m1 (3.9×10^4)	40.50	50.71	15 380	7.2	9.1	1.25
m2 (7.8×10^4)	41.25	90.00	23 922	11.4	24.9	2.18
m3 (1.5×10^5)	42.00	161.43	29 408	14.6	56.1	3.84
m4 (3.1×10^5)	44.00	304.29	46 821	24.0	165.8	6.92
m5 (6.0×10^5)	47.00	538.57	71 663	40.2	460.8	11.46
m6 (1.2×10^6)	60.75	1452.86	136 809	108.2	2588.0	23.92

Supplementary Table S12. Per-step wall time on turbulent Rayleigh–Bénard (Ra $= 10^7$, $\Gamma = 2$; main-text Fig. 3d,e). The CNN (structured finite difference, GPU) is outside the matched pair. CPU FEM is FEniCSx running the same scheme and discretization with PETSc CG + GAMG on 32 cores. Per-step cost is the least-squares slope of wall-clock time against step count once the flow is developed, at matched time step. The last column is the CPU FEM time over the GNN time. Carried to stationarity at h_{64}, the CPU FEM leg reaches $\langle \mathrm{Nu} \rangle = 14.62$ and the GNN 14.64 (production run, mass tolerance 10^{-7}).

Mesh	CNN (ms)	GNN (ms)	CPU FEM (ms)	CPU/GPU (FEM)
h_{64}	6.6	76.8	167.3	2.2
h_{128}	7.5	171.2	613.8	3.6

Heat sink. On the three-dimensional conjugate heat sink the CPU/GPU ratio rises from 14.4 to 181.2 (Supplementary Table S14). Across the 31× size range the GPU per-step cost rises 7× and the CPU cost 88×.

Supplementary Table S13. Same-discretization throughput on the body-fitted engine geometry. The θ columns give each implementation's peak interface temperature at the end of its fixed step count (200 steps on the GPU, 60 on the CPU), within 0.0002 of the converged 0.1833 of Supplementary Table S17. Doubling the CPU allocation from 16 to 32 cores moved the CPU time by −3 to +1% at every mesh. The ratio is CPU over GPU time per step. GPU: graph multigrid on one H200; CPU: FEniCSx with PETSc CG + GAMG on 32 cores. θ is the nondimensional peak interface temperature; total DOF is $4N_{P_2} + N_{P_1}$. The first row is a coarser timing-only mesh that is not in Supplementary Table S17; the second and third rows are the medium and fine meshes of that table.

P_2 DOF	total DOF	GPU (ms)	CPU (ms)	ratio	θ GPU	θ CPU
8568	35542	161.8	130.9	0.81	0.1830	0.1829
88897	367586	226.0	2612.5	11.56	0.1831	0.1830
178636	738087	241.6	5771.0	23.89	0.1831	0.1835

Supplementary Table S14. Same-discretization throughput on the 3D conjugate heat sink. Degrees of freedom are the total count $4N_{P_2} + N_{P_1}$. GPU: graph multigrid on one H200; CPU FEM: FEniCSx with PETSc CG + GAMG on 32 cores. Meshes m1 to m3 are heat-sink meshes, not the conjugate-cavity meshes of Supplementary Table S11. m2 is the coarse mesh of Supplementary Table S21; m1 and m3 appear only here.

Mesh	total DOF	GPU (ms)	CPU FEM (ms)	CPU/GPU
m1	3.92×10^5	228.9	3302.2	14.4
m2	1.98×10^6	316.8	22 609.4	71.4
m3	1.20×10^7	1598.5	289 614.5	181.2

S5.1 Preconditioner strength

Graph multigrid needs 17–42% more preconditioned-CG iterations per pressure solve than PETSc CG + GAMG at every conjugate-cavity mesh over the first 500 to 700 steps from rest (Supplementary Table S15). The gap is widest (42%) at m6, the mesh of the 23.9× speedup quoted in the main text (Supplementary Table S11). Both implementations cold-start each solve from zero, because the projection solves for the pressure increment. Both stop on the unpreconditioned residual $\|b - Ax\|/\|b\|$. The imbalance runs against the GPU, which reaches the same tolerance with more Krylov work. Normalizing to a common iteration count would widen the per-step gap. The counts compare conditioning and carry no timing. The two hierarchies are different algebraic-multigrid implementations at their library defaults. Over the whole run to the steady Nu the graph-multigrid count falls to 10.0 to 14.6 per solve; identical runs differ by about 2% in that whole-run count because the Chebyshev bound starts from an unseeded random vector. The CPU count was not measured over the whole run.

Matched tolerance alone does not show matched preconditioner strength, because two solvers can reach the same tolerance with very different Krylov work. On the test operators shipped with the code package, the same V-cycle used as a stationary iteration needs 1.5 to 2.4 times the conjugate-gradient iteration count on the conjugate-cavity and engine-bore operators and 2.8 to 14 times on the heat-sink operator (code package, experiment `stationary_vs_pcg`). A V-cycle built on a constant-coefficient operator and applied to a high-contrast one diverges as a stationary iteration at each of the five damping factors tried, 1, 0.5, 0.3, 0.1 and 0.05. All five lie far above the stability limit, which is at most 3×10^{-3} on both targets, and conjugate gradients with the same V-cycle converge (code package, experiment `stationary_mismatch`).

S5.2 The same solver on CPU and GPU

The identical graph multigrid is 4.6× slower per step on 16 CPU threads than on the GPU at the coarsest conjugate mesh. At the finest mesh, m6, it is 21.4× slower (Extended Data Table 3). That is close to the same-method 23.9× against PETSc CG + GAMG (Supple-

mentary Table S11). The two come from separate campaigns with different hierarchies and iteration counts, so they do not separate device from backend numerically. This CPU execution is not optimized, and PETSc CG + GAMG remains the CPU reference.

S5.3 Isolated pressure solve

At 1.68×10^7 pressure DOF and a shared 10^{-6} tolerance, mixed precision is 1.29× and AmgX 1.39× faster per solve than fp64 graph multigrid (Extended Data Fig. 4a). AmgX needs 2.15× less peak device memory at scale, while mixed precision needs about 44% more, storing the fp32 hierarchy beside the fp64 operator. The fp64 baseline runs to convergence, as in Rayleigh–Bénard, whereas the cavity and conjugate runs cap the cycle count. An fp32 V-cycle inside fp64 conjugate gradients converges in the fp64 iteration count, while a pure fp32 solve stalls above the 10^{-8} tolerance (code package, experiment `mixed_precision`, on the package operators).

S5.4 Pressure backend per time step

Within a whole time step the isolated-solve advantage of AmgX shrinks to 0.5–4.7% per step on the assembled-Laplacian path (Supplementary Table S16). That path covers the conjugate cavity and Rayleigh–Bénard. On Rayleigh–Bénard alone the gain is 3.1–4.7%. Mixed precision does not reduce the step time there. On the matrix-free Schur path graph multigrid is fastest on the 2D cavity and on the 3D cavity at h_{64}; at 3D h_{128} mixed precision is marginally faster. There AmgX is 3–42% slower, and mixed precision is 61–84% slower in 2D, where conversion to fp32 dominates the cheap Schur cycle. All three backends solve the same operator to 10^{-6} on one H200 in fixed-step timing runs. The Nusselt number at the last step agrees across backends to all printed digits on the 2D cavity and on the 3D cavity at h_{64}, to 2×10^{-5} at 3D h_{128}, and to 0.7% on the short

Supplementary Table S15. Preconditioned-CG iterations per pressure solve at matched tolerance. Conjugate cavity, the same anchored Laplacian and 10^{-6} true-residual criterion in both implementations, averaged over the first 504 to 709 steps from rest, both legs in one job per mesh on an exclusive node. Counts grow from 10.3 to 16.3 with graph multigrid and from 8.8 to 11.5 with GAMG over a 31× range of problem size. We make no claim as to which library's multigrid is superior.

Mesh	P_2 DOF	graph multigrid (GPU)	CG + GAMG (CPU)
m1	11 963	10.3	8.8
m2	23 899	10.9	8.9
m3	46 929	12.5	9.0
m4	94 291	11.2	9.3
m5	185 949	12.9	9.9
m6	373 073	16.3	11.5

Rayleigh–Bénard runs, which are not converged in Nu. Graph multigrid runs CUDA-graph captured on both paths. On the assembled-Laplacian path mixed precision is captured as well and AmgX is not. On the Schur path the driver's capture covers graph multigrid only, so the mixed and AmgX arms ran eagerly and the Schur-path advantage of graph multigrid includes capture.

Supplementary Table S16. Per-step wall time (ms) by pressure backend, full solver step at converged-benchmark sizes on one H200. graph multigrid: fp64, CUDA-graph captured. mixed: fp32 V-cycle inside fp64 conjugate gradients. AmgX: NVIDIA AmgX, fp64. One run per cell; a repeat of the m3 pair on the published code put AmgX 2% slower. The h_{256} Rayleigh–Bénard row is faster per step than h_{128} because its 400-step window is still in the conduction state (Nu 1.0001 at step 400), so its pressure solves take few iterations. Absolute times are not comparable with Supplementary Table S12, which uses the production driver.

problem	mesh	graph multigrid	mixed	AmgX
Rayleigh–Bénard	h_{128}	88.0	90.4	85.3
	h_{256}	65.7	65.9	62.6
conjugate cavity	m2	43.1	44.3	41.6
	m3	44.1	44.6	43.9
cavity 2D	h_{128}	14.9	27.4	15.3
	h_{256}	18.6	30.0	22.0
cavity 3D	h_{64}	169.7	195.8	210.0
	h_{128}	1238.8	1234.8	1759.2

S6 Engine cylinder

The monolithic conjugate solver of main-text Section 2.5 gives a grid-independent peak interface temperature, $\theta = 0.1833$ on three meshes, on a three-dimensional single-cylinder engine in the configuration of the ANSYS Forte tutorial [12]. A gas bore sits inside a metal head and liner. The combustion load is a prescribed cycle-averaged heat flux q'' on the chamber wall. That wall is the fluid–solid interface, so the interface carries a source (Equation (6) in Methods).

One monolithic system. The tutorial workflow is partitioned: Forte solves the reacting gas, Fluent solves the solid, and System Coupling iterates a Dirichlet–Neumann loop with under-relaxation. We solve gas and metal as one system, so there is no outer loop and no lagged coupling error.

Geometry and boundary conditions. The bore has radius $R = 0.3L$, is centred at $(x, y) = (0.5, 0.5)L$, spans $z = 0.12L$ to the fire deck at $z = 0.78L$, and sits in a metal cube of side L. The chamber wall carries q'' and no-slip for the gas, and the six outer faces are held at the coolant temperature. Conforming tetrahedral meshes resolve

the interface by element facets, with minimum dihedral angle $\approx 12°$ and none below 10° (Supplementary Table S17).

Scaling and scope. With $k_s/k_f = 10^3$ and the conduction scale $\Delta T_{\mathrm{ref}} = q'' L/k_s$, the buoyancy Rayleigh number is $\mathrm{Ra} = 10^5$ and the nondimensional wall flux is $q^* = k_s/k_f$. Because ΔT_{ref} enters Ra, these groups and the dimensional inputs cannot all be set independently for one real gas. The case is therefore a similarity problem, and its Kelvin values are only an illustrative mapping. The inputs were not calibrated to a target temperature. No validation against engine physics was performed, so agreement with operating cylinder-head temperatures is only a plausibility check. The model omits radiation, compressible in-cylinder motion, combustion chemistry and coolant-jacket flow, and the tutorial is a reference for the coupling structure only.

Result. The nondimensional peak interface temperature, which sits on the fire deck, is $\theta = 0.1833$ on all three meshes, grid-independent to four significant figures (Supplementary Table S17). Buoyancy lifts gas heated at the chamber wall to the fire deck, and the metal conducts the heat to the cooled outer faces (Extended Data Fig. 5). On these three meshes the graph-multigrid pressure solve takes $\approx$16–22 iterations per solve. The mean interior-metal temperature, an unweighted average over the metal P_2 nodes that do not touch the gas, drifts monotonically by ≈ 3 K between the two finest meshes and has not reached a plateau, so it is indicative only.

Temperatures map to Kelvin as $T = T_{\mathrm{cool}} + \theta\, q'' L/k_s$. For a cast-iron head and liner ($k_s = 55\,\mathrm{W\,m^{-1}K^{-1}}$), coolant at $T_{\mathrm{cool}} = 360\,\mathrm{K}$, $q'' = 0.5\,\mathrm{MW\,m^{-2}}$ and $L = 0.1\,\mathrm{m}$, the fire deck maps to $\approx 527\,\mathrm{K}$ and the interior-metal mean to $\approx 401\,\mathrm{K}$. The fire-deck value is of the order expected for an operating cast-iron cylinder head [13]. The implied $\approx 167\,\mathrm{K}$ rise from coolant to fire deck is large for the constant-property Boussinesq model. A dimensional prediction would also have to justify that model at this temperature difference.

Supplementary Table S17. Engine meshes and metal-temperature grid convergence. Kelvin values use $T = T_{\mathrm{cool}} + \theta\, q'' L/k_s$ with $k_s = 55\,\mathrm{W\,m^{-1}K^{-1}}$, $T_{\mathrm{cool}} = 360\,\mathrm{K}$, $q'' = 0.5\,\mathrm{MW\,m^{-2}}$ and $L = 0.1$ m. The interior-metal mean is the unweighted average over the P_2 nodes of the metal that do not touch the gas. "min dih." is the minimum tetrahedron dihedral angle, "iface facets" the number of fluid–solid interface triangles and h_Γ/R their mean edge length relative to the bore radius. Fire-deck values correspond to $\theta = 0.1833$ on every mesh.

Mesh	tets	P_2 DOF	min dih. (deg)	iface facets	h_Γ/R	fire-deck (K)	interior-metal mean (K)
coarse	27573	41036	13.7	3178	0.121	526.6	394.4
medium	61327	88897	12.0	5684	0.091	526.6	397.6
fine	125781	178636	12.0	9260	0.071	526.6	400.6

S7 Distributed execution

Eight H200 GPUs on one node carry 1.18×10^8 DOF on the conjugate heat sink and 4.7×10^8 on the engine (Supplementary Table S22). Each strong-scaling comparison holds the mesh and discretization fixed and varies only the pressure preconditioner. Block-Jacobi's pressure iterations grow with the GPU count, and a two-level coarse correction restores positive strong scaling on the transient problems.

Decomposition does not change the solution beyond solver tolerance. On the turbulent Rayleigh–Bénard cell (h_{256}, $\mathrm{Ra} = 10^7$) a two-GPU run follows the single-GPU Nusselt-number trajectory to a relative difference of 8×10^{-11} over 120 steps. On the conjugate cavity, the distributed solver run on one GPU and on two reproduces the value of the single-GPU driver, $\mathrm{Nu} = 2.7995$, to 1.4×10^{-7}. On the engine, one-GPU and multi-GPU runs agree in the flux-weighted metal temperature to a relative difference below 10^{-11}, at most 6.4×10^{-12} at two, four and eight GPUs with both preconditioners. They also reproduce the 526.6 K fire-deck temperature of Supplementary Table S17.

Degrees of freedom. In this note the steady-cavity tables count pressure degrees of freedom (DOF) and the other tables count total DOF over all fields. In two dimensions the total is about 13 times the pressure count.

Preconditioners. All comparisons are between our own preconditioners. One-level block-Jacobi applies the graph-multigrid V-cycle rank by rank, with no coupling between ranks. The steady cavity compares it with Nicolaides deflation, a one-degree-per-rank coarse space. The four transient problems compare it with a two-level preconditioner, which adds a global aggregation coarse correction solved redundantly on every rank. At one GPU block-Jacobi and two-level are the same solve.

Iteration units and the pressure cap. Transient pressure iterations are counted either per step, summed over the three Runge–Kutta substages, or per solve. Each column heading or block label says which. The distributed transient runs cap the pressure solve at 300 iterations per solve (900 per step); the conjugate-cavity strong-scaling runs cap it at 100 per solve (300 per step). A dagger (†) marks entries where pressure solves reach the cap. There the pressure is under-resolved and the time measures capped work. The two-level preconditioner converges below the cap at every point. Rayleigh–Bénard and conjugate-cavity timings are medians of three repetitions on exclusive nodes. The engine timings are medians of three and the heat-sink scaling runs single runs, each holding all eight GPUs of its node without the exclusive flag. The steady-cavity tables are medians of five runs on exclusive nodes.

S7.1 Preconditioner comparisons

Block-Jacobi's pressure iterations grow from 29.0 to 170.6 per outer step between one and eight GPUs on the fixed h_{128} steady cavity (Supplementary Table S18). The wall-clock speedups average 50 outer iterations from rest, including a first iteration that takes about 1.6 s on one GPU and up to 17 s on two to eight GPUs. On that average block-Jacobi's speedup is highest at four GPUs. From the second iteration on it reaches 1.83× on both four and eight GPUs, so it flattens rather than falls. Nicolaides deflation cuts the iterations by 29 to 35% for about 10% more time per iteration and gives the higher speedup at every multi-GPU count: 1.92× at four GPUs on the 50-iteration average, and 2.16× at four and 2.34× at eight from the second iteration on.

At a fixed load of about 1.5×10^6 pressure DOF per GPU, deflation saves a net 22 to 29% of pressure-solve time (Supplementary Table S19).

Supplementary Table S18. Strong scaling of the distributed GNN solver. Fixed 3D cavity, $\mathrm{Ra} = 10^6$, h_{128} ($\approx 1.57 \times 10^6$ pressure DOF total), one to eight GPUs. bj: block-Jacobi; Nic.: Nicolaides deflation; "it/o" is pressure iterations per SIMPLE outer step; ms/PCG-it is time per CG iteration. The wall-clock speedup t_1/t_N is on the whole SIMPLE step, averaged over 50 outer iterations from rest with the first iteration included; medians of five runs.

		it/o		ms/PCG-it		wall-clock speedup	
GPUs	pressure DOF/GPU	bj	Nic.	bj	Nic.	bj	Nic.
1	1.57M	29.0	29.0	6.29	6.29	1.00×	1.00×
2	785k	88.7	63.1	4.28	4.71	1.30×	1.47×
4	393k	121.8	82.9	3.38	3.74	1.64×	1.92×
8	196k	170.6	110.1	3.09	3.41	1.56×	1.72×

Supplementary Table S19. Weak scaling of the distributed graph-multigrid pressure solve. 3D cavity, $\mathrm{Ra} = 10^6$, $\approx 1.5 \times 10^6$ pressure DOF per GPU. "it/o" is pressure iterations per SIMPLE outer step; ms/PCG-it is time per CG iteration. "bj eff" is block-Jacobi's weak efficiency on ms/PCG-it against one GPU; "Nic. net" is the deflation's net pressure-solve saving. The per-GPU load follows the admissible cubic grids and is not constant, so "bj eff/DOF" gives the same efficiency per DOF. Abbreviations as in Supplementary Table S18.

GPUs	pressure DOF/GPU	bj it/o	Nic. it/o	bj ms/PCG-it	bj eff	bj eff/DOF	Nic. net
1	1.57M	29.0	29.0	6.30	1.00	1.00	–
2	1.51M	94.7	69.3	6.84	0.92	0.89	22%
4	1.31M	147.8	102.4	6.50	0.97	0.81	26%
8	1.50M	230.5	150.1	7.58	0.83	0.79	29%

A two-level coarse correction gives positive strong scaling on all four transient problems (Supplementary Table S20). On the Rayleigh–Bénard cell, the conjugate cavity and the heat sink it reaches 1.18 to 1.34× on eight GPUs, a parallel efficiency near 15%. Block-Jacobi's pressure iterations grow with the GPU count on every problem and reach the cap on Rayleigh–Bénard and the conjugate cavity.

Supplementary Table S20. Fixed-size strong scaling with block-Jacobi and two-level preconditioners. † marks entries where pressure solves reach the cap: 900 per step (300 per solve) on Rayleigh–Bénard and 300 per step (100 per solve) on the conjugate cavity. The monitored quantity is identical across entries: $\langle \mathrm{Nu} \rangle$ on Rayleigh–Bénard, θ_{src} on the heat sink, and the fire-deck and flux-weighted metal temperatures on the engine. Rayleigh–Bénard $\mathrm{Ra} = 10^7$, h_{1448}, 5.5×10^7 total DOF; conjugate cavity $\mathrm{Ra} = 10^6$, $T_p = 0.1$, h_{2040}, 5.4×10^7; heat sink $\mathrm{Ra} = 10^6$, 1.57×10^7; engine $\mathrm{Ra} = 10^5$, $k_s/k_f = 10^3$, 6.2×10^7.

	ms/step		speedup t_1/t_N		pressure iterations	
GPUs	block-Jac.	two-level	block-Jac.	two-level	block-Jac.	two-level
Unsteady Rayleigh–Bénard (per step)						
1	1154	1150	1.00×	1.00×	78	78
2	2032	980	0.57×	1.17×	590	118
4	2179	868	0.53×	1.32×	898†	148
8	1882	857	0.61×	1.34×	900†	161
Two-dimensional conjugate cavity (per step)						
1	900	901	1.00×	1.00×	81	80
2	1134	839	0.79×	1.07×	300†	117
4	838	771	1.07×	1.17×	300†	143
8	724	727	1.24×	1.24×	300†	159
Three-dimensional conjugate heat sink (per step)						
1	819	819	1.00×	1.00×	84	84
2	931	742	0.88×	1.10×	251	104
4	1030	692	0.80×	1.18×	338	115
8	1314	695	0.62×	1.18×	469	126
Single-cylinder engine (per solve)						
1	4016	4016	1.00×	1.00×	27	27
2	2954	2573	1.36×	1.56×	116	40
4	2350	1797	1.71×	2.24×	159	47
8	2452	1494	1.64×	2.69×	223	51

S7.2 Three-dimensional conjugate heat sink

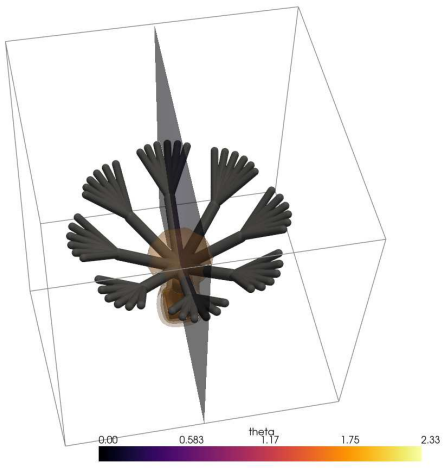
The source-averaged temperature of the heat sink converges to $\theta_{\mathrm{src}} \approx 2.36$ once the mesh resolves the conductor branches (Supplementary Table S21; temperature field in Supplementary Figure S5). The finest mesh is at the single-GPU memory ceiling. The decomposition of this three-dimensional problem exchanges two-dimensional halo faces, with an all-reduce in every Krylov iteration.

Supplementary Table S21. Grid convergence of the 3D conjugate heat-sink source temperature. Source-averaged nondimensional temperature θ_{src} at thermal steady state, $\mathrm{Ra} = 10^6$, $k_s/k_f = 100$, without calibration to a target temperature. Coarse meshes staircase the thin conductor branches and over-count the solid.

Mesh	total DOF	θ_{src}	branches resolved?
coarse	1.98×10^6	1.400	no (staircased)
fine	6.95×10^6	2.362	yes
finest	1.57×10^7	2.361	yes

Supplementary Table S22. Weak scaling of the two three-dimensional conjugate problems at the single-GPU memory ceiling. Two-level preconditioner and x-slab decomposition; the mesh grows in proportion to the GPU count. Weak efficiency is t_{ref}/t_N against two GPUs and against one; the single-GPU run exchanges no halo. Heat sink $\mathrm{Ra} = 10^6$; engine $\mathrm{Ra} = 10^5$. The load per GPU falls by 6.4% on the heat sink and 5.2% on the engine by eight GPUs, so per DOF the eight-GPU efficiencies against one GPU are 0.53 and 0.63.

GPUs	total DOF/GPU	total DOF	ms/step	weak eff. (vs 2 GPU)	weak eff. (vs 1 GPU)
3D conjugate heat sink					
1	1.57×10^7	1.57×10^7	821.4	–	1.00
2	1.50×10^7	3.01×10^7	1081.1	1.00	0.76
4	1.51×10^7	6.05×10^7	1290.0	0.84	0.64
8	1.47×10^7	1.18×10^8	1462.6	0.74	0.56
Single-cylinder engine					
1	6.16×10^7	6.16×10^7	4016	–	1.00
2	6.13×10^7	1.23×10^8	4879	1.00	0.82
4	6.02×10^7	2.41×10^8	5548	0.88	0.72
8	5.84×10^7	4.67×10^8	6023	0.81	0.67



Supplementary Figure S5. Three-dimensional conjugate heat sink. Steady temperature field at $\mathrm{Ra} = 10^6$ and $k_s/k_f = 100$ on the converged mesh. A volumetric source at the base heats the branched conductor. The conductor is drawn as a solid; the warm surfaces are temperature isosurfaces; the colour scale is clipped at the 99.5th percentile of θ. Colour: nondimensional temperature θ.

S8 Numerical methods and reproducibility

All production runs use double precision (fp64). The mixed-precision and fp32 pressure-solve tests of Supplementary Note S5 are the only exceptions. The discrete equations and exact settings below support independent reimplementation.

S8.1 Steady SIMPLE equations (cavity)

On the P_2–P_1 Taylor–Hood pair the steady cavity is advanced by a pseudo-transient SIMPLE iteration. Each outer iteration solves the relaxed momentum system for a provisional velocity $\mathbf{u}^*$,

$$\Big[\mathsf{A}(\mathbf{u}) + \tfrac{1-\alpha_U}{\alpha_U}\,\mathrm{diag}(a_P)\Big]\,\mathbf{u}^* = \tfrac{1}{\Delta t_\tau}\mathsf{M}\,\mathbf{u}^n + \tfrac{1-\alpha_U}{\alpha_U}\,\mathrm{diag}(a_P)\,\mathbf{u}^n + \mathbf{f} - \mathsf{G}\,p, \tag{S5}$$

where $\mathsf{A}(\mathbf{u}) = \frac{1}{\Delta t_\tau}\mathsf{M} + \mathrm{Pr}\,\mathsf{K} + \mathsf{N}(\mathbf{u})$. Here M and K are the mass and stiffness operators, $\mathsf{N}(\mathbf{u})$ the convective Galerkin advection operator, $\mathbf{f}$ the buoyancy load $+\mathrm{Ra}\,\mathrm{Pr}\,\mathsf{M}T$ on the vertical component and Δt_τ the pseudo-time step. The diagonal $a_P \equiv \mathrm{diag}\,\mathsf{A}$ is *unrelaxed* and includes any deferred-correction diffusion. Relaxation enters implicitly through the diagonal and the matching right-hand-side term, so the converged fixed point does not depend on α_U. With ϕ the P_2 velocity and ψ the P_1 pressure basis, $\mathsf{G}_{d,ij} = \int \phi_i\,\partial_d\psi_j$, so $\mathsf{G}^\top\mathbf{u} = -\int \psi\,(\nabla\cdot\mathbf{u})$ is the negative weak divergence. The pressure correction p' solves the Schur-complement system and corrects the velocity,

$$\mathsf{S}\,p' = \mathsf{G}^\top\mathbf{u}^*, \quad \mathsf{S} = \mathsf{G}^\top\mathrm{diag}(r_{AU})\,\mathsf{G}, \quad r_{AU} = \frac{1}{a_P}, \qquad \mathbf{u} = \mathbf{u}^* - r_{AU}\,\mathsf{G}\,p', \quad p \leftarrow p + \alpha_p\,p', \tag{S6}$$

with under-relaxation $(\alpha_U, \alpha_p, \alpha_T)$. Carrying α_U into r_{AU} would rescale p' by $1/\alpha_U$ and cancel the pressure under-relaxation at the high-Ra setting $\alpha_U = \alpha_p$. The gauge subtracts the right-hand-side mean and pins one P_1 degree of freedom, on which the graph-multigrid operator is anchored. The operator S is applied matrix-free as $\mathsf{S}v = \sum_d \mathsf{G}_d^\top\big(r_{AU}\odot\mathsf{G}_d v\big)$. The transient Rayleigh–Bénard and conjugate cases use the RK3–CN scheme of main-text Section 4.3, with the same graph-multigrid pressure solve.

S8.2 Solver configuration

Supplementary Table S23 lists the settings. No SUPG, PSPG or grad–div stabilization is used. The Taylor–Hood pair is inf-sup stable, so no pressure stabilization is needed. Deferred correction adds $\gamma_{\mathrm{mom}}\mathsf{K}$ implicitly with the matching explicit term, where $\gamma_{\mathrm{mom}} = \max(0, \mathrm{Pe}-1)\mathrm{Pr}$ and $\mathrm{Pe} = |\mathbf{u}|_{\max}h_{\min}/(2\mathrm{Pr})$. It vanishes at the fixed point but changes the convergence rate, and hence the wall-clock time.

Nusselt-number evaluation. The GNN and FEniCSx evaluate the same boundary integral of main-text Equation (7) on the P_2 temperature. The CNN uses a one-sided finite-difference wall gradient and OpenFOAM its `wallHeatFlux` function object. For Rayleigh–Bénard, $\mathrm{Nu} = \frac{1}{2}(\mathrm{Nu}_{\mathrm{bot}} + \mathrm{Nu}_{\mathrm{top}})$, time-averaged.

S8.3 Hardware and software

Supplementary Table S24 lists the platform.

S8.4 OpenFOAM stopping and inner-tolerance sensitivity

OpenFOAM stops at the identical outer iteration when its inner linear-solve tolerance moves an order of magnitude either side of the production 10^{-7} (Supplementary Table S25). An external monitor applies the shared $|\Delta\mathrm{Nu}/\mathrm{Nu}| < 10^{-5}$ test to the wall-heat-flux trace, so OpenFOAM's own solver tolerances govern only its inner GAMG and PBiCGStab solves. The converged Nu moves by $\lesssim 9 \times 10^{-5}$ relative. The exception is h_{256}, $\mathrm{Ra} = 10^4$, where the loose 10^{-6} setting sits 0.12% low and the other two both give 2.2449. Tightening to 10^{-8} costs at most 22% more wall-clock time.

S8.5 Stopping criterion: sensitivity to a combined test

A stricter combined stopping test changes the converged Nu by at most 0.05% for the GNN and not at all, to five figures, for FEniCSx (Supplementary Table S26). It adds a Nusselt tolerance of 10^{-6}, a velocity update norm below 5×10^{-3} and a divergence below 10^{-3} to the production test.

One cell stays unresolved: h_{256} at $\mathrm{Ra} = 10^4$. There FEniCSx sits 0.35% below the GNN in the production runs (2.237 in Extended Data Table 1), and the combined test leaves the gap in place. The two codes assemble bit-identical operators from the same Unified Form Language (UFL) forms, and the GPU advection matrix matches the compiled form to $\sim 10^{-14}$. Under the combined test the GNN value matches OpenFOAM (2.24488) and de Vahl Davis (2.243). At h_{128}, $\mathrm{Ra} = 10^4$ the GNN and FEniCSx values differ by only $\approx 0.05\%$. The GNN needs about 2.3 times as many SIMPLE iterations as FEniCSx takes to stop (70800 against 30760), so we suspect incomplete FEniCSx convergence. We did not run the FEniCSx cell that long. It stays an unresolved outlier and is excluded from the refinement trend.

S8.6 Data and code availability

The code package `gnn4buoyancy-mwe` (repository `multigrid-gnn`) is public under the MIT licence at `github.com/teeratornk/multigrid-gnn` (release `v1.2.2`). It holds the

minimal working example of the graph-multigrid pressure solve and runnable pressure-solve experiments with their own configuration groups. It also holds the tables and figures that left this Supplementary Information, or lost columns in the cut, with the data and code that rebuild the tables and four of the five figures, and these notes as they stood before they were shortened. The production solver source is not included. On publication, a public repository with a citable DOI will hold an archived copy of the package, the production solver configuration groups and the voxelized and body-fitted benchmark meshes. It will also hold the per-run result files, each with its resolved configuration, and the processed Nusselt-number, wall-clock and scaling tables.

Supplementary Table S23. Exact solver configuration of the production runs. Rows give the GNN setting unless they name another solver. All four solvers run steady SIMPLE on the cavity. On Rayleigh–Bénard and the conjugate cavity the GNN and FEniCSx march RK3–CN; on Rayleigh–Bénard the CNN uses an explicit Heun scheme (main-text Section 4.3).

Item	Value
Element pair	P_2–P_1 Taylor–Hood (velocity, temperature P_2; pressure P_1)
Advection	convective Galerkin $N_i\,\mathbf{u}\cdot\nabla N_j$, deferred correction
Quadrature	degree 5; no over-integration
Stabilization	none; deferred-correction diffusion $\gamma_{\mathrm{mom}}\mathsf{K}$ on the single-GPU 2D and 3D cavity runs; off in the distributed 3D cavity scaling sweeps (Supplementary Tables S18 and S19)
Mass matrix	consistent, Jacobi-PCG; rotational term uses the P_1 lumped mass
Mass-solve relative tolerance	10^{-10}: single-GPU conjugate cavity, Rayleigh–Bénard per-step pair (Supplementary Table S12), engine and heat-sink throughput (Supplementary Tables S13 and S14); 10^{-7}: single-GPU Rayleigh–Bénard production and statistics, engine and heat-sink grid-convergence runs, all distributed runs; 10^{-8}: the House benchmark. GPU mass solves are capped at 300 iterations
SIMPLE relaxation	$\alpha_U/\alpha_p/\alpha_T = 0.7/0.3/0.9$; $\alpha_U = 0.5$ at $\mathrm{Ra} \geq 10^5$, 0.3 at $\mathrm{Ra} \geq 10^6$
Pseudo-time step	$\Delta t_\tau = 10^5\, h^2/\mathrm{Pr}$, clamped $\leq 1/\mathrm{Pr}$ ($\mathrm{Ra} \geq 10^6$), $\leq 3/\mathrm{Pr}$ ($\mathrm{Ra} \geq 10^5$)
Cavity inner solves	momentum, energy, pressure rtol 10^{-6}; pressure max iter 100
Transient scheme	rotational-IPCS, RK3 advection, Crank–Nicolson diffusion
Transient Δt (RBC)	adaptive, CFL target 1.0 (high-water mark)
Transient Δt (conjugate)	fixed at start-up to $\mathrm{CFL}\cdot h/u_{\mathrm{char}}$ with $h = h_{\min}$ (conjugate cavity, engine) or the 5th-percentile fluid edge (heat sink); same in both codes (Supplementary Table S11)
CNN Δt (RBC)	explicit, $0.1\,\Delta x/\sqrt{\mathrm{Ra}\,\mathrm{Pr}}$ (2.93×10^{-7} at h_{128}, $\mathrm{Ra} = 10^7$)
Pressure solver	CG with one graph-multigrid V-cycle, rtol 10^{-6}
Pressure iteration cap	100 cavity; 50 single-GPU RBC and conjugate; 500 distributed cavity; 300 per solve, distributed transient (100 for the conjugate-cavity strong scaling). Exceptions: the House benchmark (cap 80, rtol 10^{-7}), the iteration probes of Supplementary Table S15 (cap 500), AmgX (its own cap 100), and the RK3–CN ladders at $\mathrm{Ra} = 10^4$ and 10^5 and the ARS(2,2,2) ladders of Supplementary Table S10, which used five fixed V-cycle-preconditioned CG iterations per solve
Multigrid	smoothed-aggregation AMG (PyAMG [14]) on the P_1 Laplacian, up to 10 levels
Smoother	Chebyshev degree 2, $n_{\mathrm{pre}} = n_{\mathrm{post}} = 2$, $\lambda_{\max}$ by 10 power steps, band $[\lambda_{\max}/6, \lambda_{\max}]$: single-GPU cavity, Rayleigh–Bénard and conjugate-cavity runs, distributed Rayleigh–Bénard and conjugate-cavity runs weighted Jacobi $\omega = 0.7$, 2/2: engine and heat sink (one GPU and distributed), the distributed 3D cavity scaling sweeps, the House benchmark, the manufactured-solution runs, and the $\mathrm{Ra} = 10^4$ and 10^5 RK3–CN and all ARS(2,2,2) ladders of Supplementary Table S10
Pressure gauge	P_1 DOF 0 pinned; SA near-null space constant
CUDA-graph capture	CG loop body (matvec, V-cycle) captured once and replayed on the single-GPU cavity, Rayleigh–Bénard and conjugate-cavity runs; the engine and heat sink run eagerly; the distributed CG loop is eager everywhere, with the per-rank V-cycle replayed from a captured graph on the distributed Rayleigh–Bénard and conjugate-cavity runs
Partitioning	x-slab (3D cavity, heat sink; engine with equal-cell-count slabs); y-slab (RBC periodic in x, conjugate cavity); halo width 1; NCCL point-to-point exchange
Nicolaides deflation	Z one indicator per rank, $E = Z^{\top}\mathsf{S}Z$; one fine V-cycle per rank
Two-level correction	PyAMG SA on the full P_1 Laplacian per rank; the coarse correction is one graph-multigrid V-cycle on the global aggregation operator, added to the per-rank fine V-cycle
Solid velocity (CHT)	P_2 velocity pinned to $\mathbf{0}$ on solid nodes

Supplementary Table S24. Hardware and software stack.

Component	Detail
GPU	NVIDIA H200 SXM, 141 GB HBM3e, NVLink intra-node
CPU	2× AMD EPYC 9454 (48-core), 251 GB RAM
Environment	One environment for the GPU solvers (CNN, GNN) and the CPU finite element reference: Python 3.12, PyTorch 2.5.1+cu124, CUDA 12.4, NCCL 2.21.5, PyAMG 5.3.0, DOLFINx 0.10.0, PETSc 3.25.1, MPICH (mpi4py 4.1.1). DOLFINx also assembles the GNN operators at setup
Adjoint reference	pyadjoint 2026.4.1, dolfinx-adjoint 0.2.1, checkpoint-schedules 1.0.4, on the same DOLFINx and PETSc stack
Finite-volume reference	OpenFOAM v12 (double precision), `foamRun -solver fluid`
Compiler	gcc 14.2.0
CPU cores	32 for every CPU reference (FEniCSx, OpenFOAM) on every case
Scheduler	Slurm. Exclusive node for the same-discretization throughput runs, the pressure-backend and iteration-count probes, the Rayleigh–Bénard per-step pair, the pyadjoint comparison and the distributed Rayleigh–Bénard, conjugate-cavity and steady-cavity scaling. The heat-sink and engine scaling runs held all eight GPUs of a node without the exclusive flag. The cavity wall-clock runs, the conjugate-cavity runs to a steady Nu and the torch-CPU runs used shared nodes

Supplementary Table S25. OpenFOAM converged Nu and cost versus inner linear-solve tolerance. 2D differentially heated cavity, one deterministic run per cell. The last column is the wall-clock ratio of the 10^{-8} and 10^{-6} runs. The outer stop is the shared $|\Delta\mathrm{Nu}/\mathrm{Nu}| < 10^{-5}$ test, and 10^{-7} is the production setting.

Mesh	Ra	Nu (10^{-6})	Nu (10^{-7})	Nu (10^{-8})	outer iters	wall $10^{-8}/10^{-6}$
h_{64}	10^4	2.2493	2.2494	2.2495	710	1.14
h_{64}	10^6	9.0546	9.0553	9.0554	710	1.09
h_{256}	10^4	2.2422	2.2449	2.2449	4760	1.22
h_{256}	10^6	8.8388	8.8392	8.8393	4760	1.16

Supplementary Table S26. Stopping-criterion sensitivity. Converged Nu of the two P_2–P_1 solvers under the Nusselt-only and combined tests, on representative cavity cells. The combined test adds a Nusselt tolerance of 10^{-6}, a velocity update norm below 5×10^{-3} and a divergence below 10^{-3} to the production test. Δ is the change; r_u is the velocity update norm at the Nusselt-only stop (≈ 0 is below printed precision).

Cell	Solver	Nu (Nu-only)	r_u at stop	Nu (combined)	Δ (%)
h_{128}, Ra=10^5	GNN	4.5218	2.7×10^{-3}	4.5220	0.004
h_{128}, Ra=10^5	FEniCSx	4.5221	≈ 0	4.5221	0.000
h_{256}, Ra=10^4	GNN	2.2446	2.0×10^{-4}	2.2449	0.014
h_{256}, Ra=10^4	FEniCSx	2.2366	≈ 0	2.2366	0.000
h_{256}, Ra=10^5	GNN	4.5187	7.2×10^{-3}	4.5211	0.052

Supplementary References

[1] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi: 10.1002/nme.2579.

[2] Igor A. Baratta, Joseph P. Dean, Jørgen S. Dokken, Michal Habera, Jack S. Hale, Chris N. Richardson, Marie E. Rognes, Matthew W. Scroggs, Nathan Sime, and Garth N. Wells. DOLFINx: The next generation FEniCS problem solving environment. Preprint, Zenodo, 2023. doi: 10.5281/zenodo.10447666.

[3] Sebastian K. Mitusch, Simon W. Funke, and Jørgen S. Dokken. dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake. *Journal of Open Source Software*, 4(38):1292, 2019. doi: 10.21105/joss.01292.

[4] E. Tric, G. Labrosse, and M. Betrouni. A first incursion into the 3D structure of natural convection of air in a differentially heated cubic cavity, from accurate numerical solutions. *International Journal of Heat and Mass Transfer*, 43(21): 4043–4056, 2000. doi: 10.1016/S0017-9310(00)00037-5.

[5] G. de Vahl Davis. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3): 249–264, 1983. doi: 10.1002/fld.1650030305.

[6] P. Le Quéré. Accurate solutions to the square thermally driven cavity at high Rayleigh number. *Computers & Fluids*, 20(1):29–41, 1991. doi: 10.1016/0045-7930(91)90025-D.

[7] Hans Johnston and Charles R. Doering. Comparison of turbulent thermal convection between conditions of constant temperature and constant flux. *Physical Review Letters*, 102(6):064501, 2009. doi: 10.1103/PhysRevLett.102.064501.

[8] H. Flyvbjerg and H. G. Petersen. Error estimates on averages of correlated data. *The Journal of Chemical Physics*, 91(1):461–466, 1989. doi: 10.1063/1.457480.

[9] Erwin P. van der Poel, Richard J. A. M. Stevens, and Detlef Lohse. Comparison between two- and three-dimensional Rayleigh–Bénard convection. *Journal of Fluid Mechanics*, 736:177–194, 2013. doi: 10.1017/jfm.2013.488.

[10] J. M. House, C. Beckermann, and T. F. Smith. Effect of a centered conducting body on natural convection heat transfer in an enclosure. *Numerical Heat Transfer, Part A: Applications*, 18(2):213–225, 1990. doi: 10.1080/10407789008944791.

[11] M. Khatamifar, W. Lin, and L. Dong. Transient conjugate natural convection heat transfer in a differentially-heated square cavity with a partition of finite thickness and thermal conductivity. *Case Studies in Thermal Engineering*, 25:100952, 2021. doi: 10.1016/j.csite.2021.100952.

[12] ANSYS, Inc. Conjugate heat transfer simulation of a single-cylinder internal combustion engine. In *Ansys Forte Tutorials*, Release 2025 R1, chapter 24, 2025. Accessed via the ANSYS Help system, 26 August 2026.

[13] J. B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill Education, New York, 2nd edition, 2018. Ch. 12, heat transfer in engines.

[14] Nathan Bell, Luke N. Olson, Jacob Schroder, and Ben Southworth. PyAMG: Algebraic multigrid solvers in Python. *Journal of Open Source Software*, 8(87): 5495, 2023. doi: 10.21105/joss.05495.