The deal.II Library, Version 9.7

Daniel Arndt[1*], Wolfgang Bangerth[2,3], Maximilian Bergbauer[4], Bruno Blais[5], Marc Fehling[6], Rene Gassmöller[7], Timo Heister[8], Luca Heltai[9], Martin Kronbichler[10], Matthias Maier[11], Peter Munch[12], Sam Scheuerman[2], Bruno Turcksin[1*], Siarhei Uzunbajakau[13], David Wells[14], and Michał Wichrowski[15]

[1]Computational Coupled Physics Group, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd., TN 37831, USA. arndtd/turcksinbr@ornl.gov

[2]Department of Mathematics, Colorado State University, Fort Collins, CO 80523, USA. bangerth/sam.scheuerman@colostate.edu

[3]Department of Geosciences, Colorado State University, Fort Collins, CO 80523, USA.

[4]Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching bei München, Germany. maximilian.bergbauer@tum.de

[5]Chemical Engineering High-performance Analysis, Optimization and Simulation (CHAOS) laboratory, Department of Chemical Engineering, Polytechnique Montréal, PO Box 6079, Stn Centre-Ville, Montréal, Québec, Canada, H3C 3A7. bruno.blais@polymtl.ca

[6]Department of Mathematical Analysis, Faculty of Mathematics and Physics, Charles University, Sokolovská 49/83, 186 75 Prague 8, Czech Republic. marc.fehling@matfyz.cuni.cz

[7]GEOMAR Helmholtz Centre for Ocean Research Kiel, 24148 Kiel, Germany

[8]School of Mathematical and Statistical Sciences, Clemson University, Clemson, SC, 29634, USA. heister@clemson.edu

[9]Department of Mathematics, University of Pisa, Via Buonarroti 1/c, 56127 Pisa, Italy. luca.heltai@unipi.it

[10]Faculty of Mathematics, Ruhr University Bochum, Universitätsstr. 150, 44780 Bochum, Germany. martin.kronbichler@rub.de

[11]Department of Mathematics, Texas A&M University, 3368 TAMU, College Station, TX 77845, USA. maier@math.tamu.edu

[12]Institute of Mathematics, Technical University of Berlin, Germany. muench@math.tu-berlin.de

[13]CEM Books, Rotterdam, The Netherlands. info@cembooks.nl

[14]Department of Mathematics, University of North Carolina, Chapel Hill, NC 27516, USA. drwells@email.unc.edu

[15]Interdisciplinary Center for Scientific Computing, Heidelberg University, Heidelberg, Germany. mt.wichrowsk@uw.edu.pl

Abstract: This paper provides an overview of the new features of the finite element library deal.II, version 9.7.

1 Overview

`deal.II` version 9.7.0 was released July 22, 2025. This paper provides an overview of the new features of this release and serves as a citable reference for the `deal.II` software library version 9.7. `deal.II` is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the terms of the *GNU Lesser General Public License* (LGPL). The `deal.II` project is in the process of relicensing the library under the terms of the *Apache License 2.0 with LLVM Exception*. Downloads are available at https://www.dealii.org/ and https://github.com/dealii/dealii.

The major changes of this release are:

- `MappingP1`, a more efficient mapping for simplex elements (Section 2.1);
- Updated support for transferring solution vectors during mesh refinement (Section 2.2);
- Improved matrix-free support for multi-component systems as well as improved portability between GPUs via `Portable::MatrixFree` (Section 2.3);
- Support for threading via TaskFlow, as well as support for several new external libraries and improved support for Kokkos and ArborX (Section 2.4);
- `deal.II` can now build C++20 modules (Section 2.5);
- Three new tutorial programs and a new code gallery program (Section 2.6).

While all of these major changes are discussed in detail in Section 2, there are a number of other noteworthy changes in the current `deal.II` release, which we briefly outline in the remainder of this section:

- MPGMRES is a variant of the widely used generalized minimal residual method (GMRES) [57], distinguished by the support for multiple preconditioners (thus, *MP*GMRES) during the course of the solution process [32]. In our implementation in the `SolverMPGMRES` class, the use of N preconditioners leads to the construction of multiple Krylov spaces via N-variate, non-commuting polynomials of the preconditioners and the system matrix applied to a residual vector. `SolverMPGMRES` supports two strategies: (i) a full variant that constructs all possible combinations of preconditioner application to the initial residual r, which for two preconditioners P_1, P_2 looks as

  $$r, P_1 r, P_2 r, P_1 A P_1 r, P_2 A P_1 r, P_1 A P_2 r, P_2 A P_2 r, P_1 A P_1 A P_1 r, P_2 A P_1 A P_1 r, \ldots, P_2 A P_2 A P_2 r, \ldots;$$

  and (ii) a truncated version where no combinations of preconditioners are considered, i.e.

  $$r, P_1 r, P_2 r, P_1 A P_1 r, P_2 A P_2 r, P_1 A P_1 A P_1 r, P_2 A P_2 A P_2 r, \ldots.$$

  The new solver is beneficial when two different preconditioners can help to more quickly span the space in which the solution is expected.

- A common operation in finite element codes is to ask whether a specific shape function belongs to a particular solution variable. In the past, this was implemented by using the `FiniteElement::system_to_component_index()` function that returns, among other information, which vector component a shape function belongs to. One then had to test whether this is the vector component (or one of the vector components) that corresponds to a specific variable. The new `FiniteElement::shape_function_belongs_to()` makes this easier: It takes as argument a `FiniteElementExtractor` that corresponds to a scalar, vector-valued, or tensor-valued set of solution variables, and directly returns `true` or `false`.

- Most of the functions in the `GridGenerator` namespace correspond to simple geometries (such as cubes, spheres, and other shapes with analytical formulas) discretized with hypercubes. The function `GridGenerator::convert_hypercube_to_simplex_mesh()` converts such discretizations to instead use simplices. That function now supports anisotropic splits, including splitting quadrilaterals into two triangles and hexahedra into six tetrahedra, in addition to the standard isotropic splits. These anisotropic splits significantly reduce the total number of cells and are useful in contexts where one would like to use as few cells as possible.

- The build system now supports interprocedural and link-time optimization (LTO). This can be enabled with the CMake configuration option `-DDEAL_II_WITH_LTO=ON`. This option enables whole-program optimization at link-time and can make executables significantly faster.

- `GridIn::read_vtk()` now loads cell-centered data vectors stored in the input file and makes them available via `GridIn::get_cell_data()`.

- Sparsity patterns for matrices built with `FE_Q_iso_Q1` elements now correctly reflect the inherited sparsity on each cell, making the usage of this element usable as a preconditioner for a higher order problems (this is known as low order rediscretization, see also [53]).

- The assembly of ghost penalty stabilization matrices, which appear in CutFEM, can now be done via `TensorProductMatrixCreator::create_1d_ghost_penalty_matrix()`. For Cartesian cells, the local penalty matrix can be constructed by exploiting the tensor-product structure of the basis functions. The ghost penalty operator is expressed as a Kronecker product of precomputed one-dimensional mass and penalty matrices, which avoids the evaluation of high-order derivatives [67]. The new functionality also handles the necessary lexicographical renumbering of degrees of freedom across adjacent cells via the `FETools::cell_to_face_patch()` function.

The changelog – listing more than 120 new features and bugfixes – contains a complete record of all changes; see [45].

2 Major changes to the library

This release of `deal.II` contains a number of large and significant changes, which we will discuss in this section.

2.1 The `MappingP1` class

Classes derived from the `Mapping` base class implement the transformation from a reference cell (such as the unit square or reference triangle) to a concrete cell that is part of a mesh. `deal.II` has supported simplex and mixed meshes since the 9.3 release [8]. That release included `MappingFE`, a `Mapping` class that uses the shape functions of a finite element to implement this transformation; many common mappings can be written in this way, and the class works with a variety of different elements by delegating most computations to a `FiniteElement` object. For example, the common bi-/trilinear mappings for quadrilaterals and hexahedra, as well as the linear mappings on triangles and simplices, can be implemented using `MappingFE` along with the `FE_Q` or `FE_SimplexP` classes.

The current release includes a new `MappingP1` class, which contains an optimized implementation of the standard affine simplex mapping (i.e., the mapping from the unit/reference cell to the real cell with a constant Jacobian on each cell). This implementation is about six times faster than the generic one implemented in `MappingFE`. In addition, this `Mapping` is now the default linear mapping returned by `ReferenceCell::get_default_linear_mapping()`, so all applications which

use that function for obtaining a `ReferenceCell`-independent mapping will automatically use the more performant class.

2.2 Updates to the `SolutionTransfer` class

This release extends the `SolutionTransfer` class to facilitate the transfer of solution vectors from one mesh to another in an adaptive mesh refinement loop, now to all triangulation classes that support mesh refinement. Consequently, it has subsumed the functionality of the existing `parallel::distributed::SolutionTransfer` class, which is no longer necessary and is now deprecated.

In the process of this update, we have also cleaned up these classes' interfaces. In particular, some member functions, such as `execute_pure_refinement()`, have been removed from the new implementation because they provided functionality for only rarely used cases, but substantially complicated the internal design of these classes. To maintain prolonged support for these edge cases and to ensure a smooth transition to the updated interface, the previous implementation remains available in the deprecated `Legacy::SolutionTransfer` class, which is planned for removal in a future release.

2.3 Matrix-free functionality

The current release includes several extensions, major bug fixes, and optimizations of the matrix-free infrastructure and related multigrid functionality, such as matrix-free geometric and polynomial coarsening. The most significant advances were made for the `Kokkos`-based "portable" infrastructure described below. In addition, improvements where made to several areas:

- The `MatrixFreeTools::compute_matrix()` and `MatrixFreeTools::compute_diagonal()` functions to compute the matrix or only its diagonal from a matrix-free operator were extended towards multi-component systems, the DG context (and, in general, to operators with face integrals), and mixed elements.
- We added support for additional ghost cells in the matrix-free setup to support vector access to a wider range of operations, such as MPI parallelization of multigrid smoothers based on vertex patches [68].
- We implemented functionality for identifying degrees of freedom in lexicographic ordering on patches of cells.
- Support for additional matrices in the `TensorProductMatrixCreator` class, supporting fast diagonalization-based inverses, was added.

In a broader context, the `Portable::MatrixFree` class utilizes `Kokkos` to provide support for GPUs from Nvidia, AMD, and Intel. In this release, we have added support for finite elements with multiple components, i.e., "vector-valued" finite elements. We have also added support for finite elements where the number of quadrature points per direction is different from the size of the one-dimensional basis (e.g., polynomial degree plus one), thus supporting more cases of evaluation. In addition to the extended functionality, several performance optimizations to the evaluation routines have been made, which are particularly visible on recent Nvidia GPUs.

To ensure more uniform handling of data structures necessary for storing temporary results in matrix-free evaluations, such as sum-factorization algorithms, we have introduced two new classes: `Portable::DeviceVector` and `Portable::DeviceBlockVector`. The first of these is an alias for `Kokkos::View` that replaces the raw pointer previously used to access data on the GPU. The class `Portable::DeviceBlockVector` extends `Portable::DeviceVector` in a manner similar to the way `BlockVector` extends `Vector` on the CPU.

Finally, we have streamlined the interface of the functor invoked by the GPU kernel. In the past, this interface included a cell index, `Portable::MatrixFree::Data` (which includes precomputed data such as quadrature points, shape functions, and other information that needs to be transferred to the GPU), and `Portable::MatrixFree::SharedData` (which contained scratch memory used for computation). Now, all this information has been consolidated into a single new class `Portable::MatrixFree::Data`, greatly simplifying the interface.

2.4 Updates of `deal.II`'s interfaces to external libraries

`deal.II` relies on external software packages for many operations that are not within the core functionality of a finite element package – see Section 3 for a complete list. As in every release, we have continued to expand and revise these interfaces. The following sub-sections outline these updates.

2.4.1 Interfaces to the Threading Building Blocks and to Taskflow A large number of operations within `deal.II` are parallelized, including through mechanisms like WorkStream [65] that are also available to and useful in application codes. For more than 15 years, since the 6.3 release, `deal.II` has used the Intel Threading Building Blocks (TBB) [55] as the underlying library to provide task- and pipeline-based parallelism. However, the TBB has downsides that primarily follow from its use of assembler mnemonics and the resulting difficulties with supporting all platforms and compilers. As a consequence, starting with the current release, `deal.II` builds on the Taskflow library [37] that is entirely implemented in C++, is header-only, and provides an interface that matches well with current C++ language standards. `deal.II` bundles the Taskflow 3.10 release, and now no longer bundles the TBB, though the TBB interfaces are still available if a TBB installation is found on a system.

In Figure 1 we compare the time for assembling the temperature matrix and temperature right-hand side in the step-32 tutorial program, like it was done in [65]. The test was performed on a 3d mesh with 393,216 cells and 3,195,010 temperature degrees of freedom using a Q_2 element. The tests are run on AMD EPYC 9654P with 96 cores (and 192 threads). One can clearly see that running all copy operations sequentially limits parallel scalability (to less than 10 threads on the left, where the copy operation is more expensive than the right), regardless of whether one uses the TBB or Taskflow. On the other hand, first "coloring" all cells to avoid conflicts in the copy operation (labeled as "colored" in the figure) eliminates this bottleneck. It is clearly visible that the Taskflow implementation is better or the same in all situations.

2.4.2 `Kokkos`, a performance portability library `Kokkos` [64] is a C++ library that makes it easy to write codes that can portably run on CPUs and GPUs of different vendors, using their respective underlying programming models. The `deal.II` 9.5 release [7] made `Kokkos` a required dependency. The current release removes the deprecated `CUDAWrappers` namespace as well as all `CUDA`-related macros in favor of `Kokkos` equivalents. The 9.7 release now bundles `Kokkos` 4.5.1 but the minimum version required is still 3.4.

2.4.3 MUMPS, a multifrontal parallel direct solver MUMPS (MUltifrontal Massively Parallel sparse direct Solver) is a widely-used software library designed for efficiently solving large sparse linear systems on parallel architectures [3].

In the 9.7 release, we have significantly enhanced integration with MUMPS, without requiring it to be installed through PETSc or Trilinos. Improvements include a simplified user interface for direct solvers, mimicking the existing interface we provide for UMFPACK, and enhanced performance optimizations that leverage recent MUMPS capabilities. Specifically, the deal.II wrappers now transparently support parallel matrix factorization, out-of-core computations, block low-rank approximations, and GPU support.

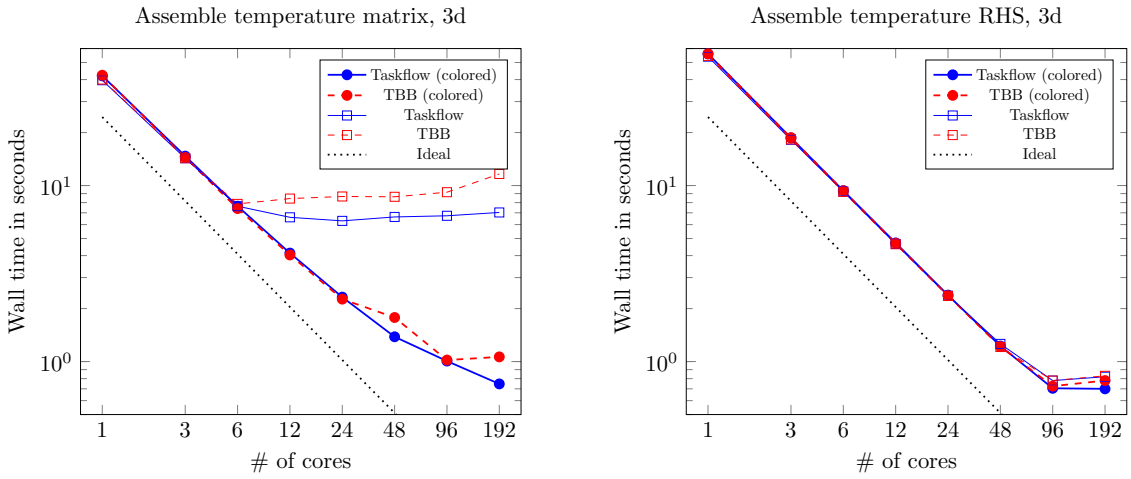

Figure 1: Assembly times in a modified step-32 (left: temperature matrix assembly, right: temperature right-hand side assembly) comparing TBB and Taskflow with and without coloring. These figures correspond to the middle row of [65, Fig. 4]. Lines with squares match "Implementation 2" in [65]; the lines with circles labeled "colored" match "Implementation 3" therein.

2.4.4 PSBLAS, a parallel sparse linear algebra library PSBLAS is a parallel sparse linear algebra library that provides basic operations for sparse matrices and vectors [26].

The library is specifically designed to leverage modern parallel computing architectures, including both CPU and GPU systems, making it well-suited for exascale computing environments. Release 9.7 includes preliminary support for configuring with PSBLAS, allowing users to take advantage of its capabilities for specific linear algebra tasks. This support is still in its early stages, and the library is not yet fully integrated into the `deal.II` ecosystem.

2.4.5 Magic Enum, a static reflection library for enums Magic Enum (alternatively known via its in-code spelling `magic_enum`) is a modern, header-only C++17 library designed to provide static reflection capabilities for enums without the need for macros or additional boilerplate code. It greatly simplifies working with enum types by enabling conversions between enum values and strings, integer values, and various other operations. Version 9.7 uses `magic_enum` to provide type safe and efficient conversions between enum values and their string representations in the `ParameterHandler` class. An example usage of this class would be:

C++ code

```
#include <deal.II/base/parameter_handler.prm>

enum class SolverType {
  DirectUMFPACK,
  DirectMUMPS,
  Iterative
};

SolverType solver_type = SolverType::DirectUMFPACK;
ParameterHandler prm;
prm.add_parameter("solver_type", solver_type);
prm.parse_input("input.prm");
```

This allows the user to specify in the parameter file `input.prm`, e.g., the type of solver to use:

```
set solver_type = DirectUMFPACK
```

The library also understands boolean operations between enums, allowing the user to specify, for example:

C++ code

```
#include <deal.II/base/patterns.h>
#include <deal.II/fe/fe_values.h>

// This is one of the deal.II enums used for the FEValues class
UpdateFlags flags = update_values;

// The following will output "update_values"
std::cout << Patterns::Tools::to_string(flags)
          << std::endl;

// While the following is equivalent to writing
// flags = update_values|update_gradients;
flags = Patterns::Tools::to_value<UpdateFlags>(
  "update_values|update_gradients");

// Allowing one to use
ParameterHandler prm;
prm.add_parameter("update_flags", flags);
prm.parse_input("input.prm");
```

In this case, the user can specify in the parameter file `input.prm` the flags to pass to an `FEValues` object:

```
set update_flags = update_values|update_gradients
```

2.4.6 ArborX, a performance-portable geometric search library ArborX [54] is a library for geometric search on CPUs and GPUs. The new ArborX 2.X series is not backward compatible with the older 1.X series. Consequently, this incompatibility required an extensive rewrite of our wrappers to support ArborX 2.X. As part of the rewrite, the template parameters had to be changed. When using the wrappers with ArborX 1.X, `ArborX::BVH` and `ArborX::DistributedTree` are templated on both the dimension and the number type. When using the wrappers with ArborX 2.X, these classes are templated on the geometric objects used for the leaf nodes of the `ArborX::BVH` type.

2.5 Work towards C++20 modules

`deal.II` currently requires a compiler that understands C++17, but some additional features are enabled if a compiler supports the C++20 standard. C++20 also introduces "modules", a way by which software packages can export their public interface without relying on the mechanism of textual inclusion of header files that C++ has inherited from C. The end goal of C++20 modules is that one can replace the use of long lists of statements such as

```
#include <deal.II/grid/tria.h>
```

by a simple

```
import dealii;
```

A lot of effort – perhaps 6 weeks of full-time work, and nearly 200 pull requests – has gone into evaluating whether `deal.II` can be built using modules (in addition to header files, which will need to be provided for a long time to come for backward compatibility). A detailed accounting of this effort can be found in [12]. The majority of this work addressed features of the `deal.II` code base, accumulated over its more than 25 year history, that were acceptable in a header-based system but not in a module-based system. To name just two of many examples: (i) there can be cycles of header files that mutually `#include` each other, but this is not allowed for modules; and

(ii) it is allowed to declare functions in header files as `static` or in anonymous namespaces, but this no longer works with modules. In both cases, while allowed, the existing code was brittle, inefficient, or hard to understand. Most of the pull requests for this project therefore simply cleaned up our code base, without having to introduce any incompatibilities.

As shown in [12], the use of C++20 modules reduces compile times for the library itself, though the evidence is mixed for tutorial programs or applications that build on `deal.II`. Given that at the moment few systems have compilers, CMake versions, and build systems that are sufficiently new to support module builds, it will be several years before this feature will become widely usable. However, the infrastructure for it is now in place, and one can build the current release into a C++20 module by providing CMake with the flag `-DDEAL_II_WITH_CXX20_MODULE=ON`.

2.6 New and improved tutorials and code gallery programs

Many of the `deal.II` tutorial programs were revised in a variety of ways as part of this release: Around 125 of the more than 1500 (non-merge) commits that went into this release touched the tutorial. In addition, there are a number of new tutorial programs:

- `step-93` demonstrates how one can implement problems in which some of the variables to be solved for are not degrees of freedom of a finite element field (i.e., they are "non-local" degrees of freedom, because they have no associated node functional that is tied to a cell or a patch of cells). Specifically, the program solves an optimization problem in which some of the variables are scalars that multiply certain source terms. `step-93` was written by Sam Scheuerman.

- `step-95` extends `step-85` and illustrates the use of matrix-free methods to solve problems with embedded boundaries using the CutFEM method [16]. It shows the interplay between the matrix-free infrastructure for standard quadrature rules (`FEEvaluation` and `MatrixFree`) and the infrastructure for unstructured quadrature (`FEPointEvaluation` and `NonMatching::MappingInfo`) while using explicit `SIMD` vectorization. The tutorial implements both continuous and discontinuous Galerkin methods in 2D and 3D. `step-95` was written by Maximilian Bergbauer, with help by Martin Kronbichler and Peter Munch.

- `step-97` is a program that solves a curl-curl problem from electromagnetics – more specifically, from magnetostatics. The program shows a formulation in which two curl-curl problems are solved consecutively to model the magnetic field that results from the current in an electrically conducting coil. `step-97` was written by Siarhei Uzunbajakau.

In addition, there is one new program in the code gallery (a collection of user-contributed programs that often solve more complicated problems than tutorial programs, and that are intended as starting points for further research rather than as teaching tools):

- *"Phase field fracture model in 3D"*, contributed by Wasim Niyaz Munshi, Chandrasekhar Annavarapu, Wolfgang Bangerth, and Marc Fehling. The program solves a formulation for fracture propagation in which one uses a separate field that tracks the "damage" to the material that has resulted from deformation, with the damage weakening the elastic properties of the deforming solid.

2.7 Incompatible changes

The 9.7 release includes around 25 incompatible changes; see [45]. Many of these incompatibilities remove previously deprecated functions or classes, or require now widely available but newer versions of external dependencies than previous `deal.II` versions. Others change internal interfaces that are not usually used in external applications. The following are worth mentioning since they are more broadly visible:

– The classes `Subscriptor` and `SmartPointer` have been renamed to `EnableObserverPointer` and `ObserverPointer`. Many core classes, such as `DoFHandler` and `Triangulation`, inherit from `EnableObserverPointer` to enable other classes to store `ObserverPointer`s pointing to them which, if dereferenced after the pointed-to object goes out of scope, will signal an error that the pointer is now "dangling". This functionality has been in `deal.II` since 1998. Since the term "smart pointers" nowadays typically refers to the `std::unique_ptr` or `std::shared_ptr` classes, these names were updated to avoid this conflict in terminology. The new names are based on the proposal for `std::observer_ptr` [19] and `std::enable_shared_from_this`.

– In order to support modules (see Section 2.5), a significant number of headers had to be split up to prevent circular inclusions and other minor incompatibilities. For example, `deal.II/grid/tria.h` no longer contains declarations of `CellData` or `SubCellData`. Those declarations have instead been moved to `deal.II/grid/cell_data.h`. In cases where this leads to errors, the situation is easily (and backward compatibly) fixed by adding an explicit `#include` statement to the user program for the file that is necessary to access declarations the compiler now no longer sees.

– As mentioned in Section 2.4.2, the current release removes the deprecated `CUDAWrappers` namespace as well as all `CUDA`-related macros in favor of `Kokkos` equivalents.

– We have continued unifying the various implementations of object orientations in the library, i.e., the description of how cells, faces, or edges are oriented relative to other such objects. While the three booleans 'orientation', 'rotation', and 'flip' have maintained their original definitions (i.e., they have default values of `true`, `false`, and `false`), the default value of the combined orientation (available via `TriaAccessor::combined_face_orientation()` and `TriaAccessor::line_orientation()`) is now 0 rather than 1. In addition, the return type of these functions is now `types::geometric_orientation`, which is a `typedef` for an 8-bit integral type.

– The `step-52` tutorial was removed. This program showed the use of time stepping functionality in `deal.II`, but this functionality was rudimentary compared to what packages such as SUNDIALS or PETSc TS offer. As a consequence, `step-52` did not show the advanced techniques we would like to promote (e.g., time stepping error control and automatic time step size choice), and it is now superseded by `step-86` (which builds on PETSc TS).

– The `parallel::distributed::SolutionTransfer` class has been merged with the class `SolutionTransfer`, with the former now being deprecated. In the process of cleaning up the interface, we have also removed some rarely used member functions of the latter that prevented us from unifying the two classes. For more information see Section 2.2.

– The `MatrixOut` class, which creates graphical representations of matrices, now defaults to creating sparse output, only showing nonzero entries. This vastly increases the size of matrices that can be visualized. Creating dense output remains possible through a flag in the class's `AdditionalData` structure.

– The `FiniteElement` class has a number of functions that allow querying properties of implementations in derived classes. These include asking whether a derived class is able to provide "(generalized) support points", i.e., whether the element defines its shape functions via nodal interpolation (or via quadrature-based integrals) and can return an array of these points. These functions returned `false` for the `FE_Nothing` element – an element that has no degrees of freedom and consequently no shape functions. This answer was not wrong, but missed the point: Codes using these functions want to know whether an element is able to provide arrays with these support points, and for `FE_Nothing` the answer should be "yes" (i.e., `true`): The class *can* provide such an array, it will just be empty.

3 How to cite deal.II

In order to justify the work the developers of deal.II put into this software, we ask that papers using the library reference one of the deal.II papers. This helps us justify the effort we put into this library.

There are various ways to reference deal.II. To acknowledge the use of the current version of the library, **please reference the present document**. For up-to-date information and a bibtex entry see

https://www.dealii.org/publications.html

The original deal.II paper containing an overview of its architecture is [14], and a more recent publication documenting deal.II's design decisions is available as [9]. If you rely on specific features of the library, please consider citing any of the following:

- For geometric multigrid: [40, 38, 21, 50];
- For distributed parallel computing: [13];
- For *hp*-adaptivity: [15, 25];
- For partition-of-unity (PUM) and finite element enrichment methods: [23];
- For matrix-free and fast assembly techniques: [42, 43];
- For computations on lower-dimensional manifolds: [24];
- For curved geometry representations and manifolds: [35];
- For integration with CAD files and tools: [36];
- For boundary element computations: [31];
- For the LinearOperator and PackagedOperation facilities: [47, 48];
- For uses of the WorkStream interface: [65];
- For uses of the ParameterAcceptor concept, the MeshWorker::ScratchData base class, and the ParsedConvergenceTable class: [58];
- For uses of the particle functionality in deal.II: [29].
- For the design of the video lectures and how they can be used in teaching: [69].

deal.II can interface with many other libraries:

- ADOL-C [33]
- ArborX [54]
- ARPACK [44]
- Assimp [59]
- BLAS, LAPACK [4]
- Boost [18]
- CGAL [61]
- Gmsh [30]
- GSL [27, 34]
- Ginkgo [5, 6]
- HDF5 [62]
- Kokkos [64]
- Magic Enum [46]
- METIS [41]
- MUMPS [3, 2]
- muparser [51]
- OpenCASCADE [52]
- p4est [20]
- PETSc [10, 11]
- PSBLAS [26]
- ROL [39]
- ScaLAPACK [17]
- SLEPc [56]
- SUNDIALS [28]
- SymEngine [60]
- Taskflow [37]
- TBB [55]
- Trilinos [49, 63]
- UMFPACK [22]
- VTK [66]
- zlib [70]

Please consider citing the appropriate references if you use interfaces to these libraries.

The two previous releases of deal.II can be cited as [7, 1].

4 Acknowledgments

deal.II is a worldwide project with dozens of contributors around the globe. Other than the authors of this paper, the following people contributed code to this release:

> Pasquale Africa, Henry Arhin, Alfredo Buttari, Bruna Campos, Nicholas Cantrell, Xiaoming Cao, Chayapol Chaoveeraprasit, Jerett Cherry, Dario Coscia, John Coughlin, Nikita Daniliuk, Crystal Farris, Marco Feder, Emmanuel Ferdman, Federico Fernandez, Olivier Gaboriault, Mohamad Ghadban, Mahdi Gharehbaygloo, Robin Görmer, Davit Gyulamiryan, Lóránt Hadnagy, Fernando Herrera, Robin Hiniborch, Quang Hoang, Jordan Hoffart, Sascha Hofstetter, Yimin Jin, Yann Jobic, Sean Johnson, Vaishnavi Kale, Sebastian Kinnewig, Andreas Koch, Jason Landini, Zhou Lei, Yingli Li, Nils Margenberg, Oreste Marquis, Ryan Moulday, Nils Much, Tileuzhan Mukhamet, Wasim Niyaz Munshi, Natalia Nebulishvili, Luz Paz, David Pecoraro, Davide Polverino, Sanjeeb Poudel, Guilhem Poy, Laura Prieto Saavedra, Sebastian Proell, Andreas Ritthaler, Mayank Sabharwal, Magdalena Schreter, Richard Schussnig, Kyle Schwiebert, Marc Secanell, Qingyuan Shi, Wyatt Smith, Simon Sticko, Dominik Still, Edward Terrell, Jan Philipp Thiele, Xiaochuan Tian, Mikael Vaillant, Vinayak Vijay, Stephan Voss, Simon Wiesheier, Yi-Yung Yang.

Their contributions are much appreciated – this project lives by its community of users and contributors!

This release contains patches written in and contributed from Antarctica. As a consequence, deal.II now consists of work written on all seven continents.

deal.II and its developers are financially supported through a variety of funding sources:

D. Arndt and B. Turcksin: Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy and supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program, under contract number DE-AC05-00OR22725.

W. Bangerth was partially supported by the National Science Foundation under awards OAC-1835673, EAR-1925595, and OAC-2410847.

W. Bangerth, T. Heister, and R. Gassmöller were partially supported by the Computational Infrastructure for Geodynamics initiative (CIG), through the National Science Foundation (NSF) under Award No. EAR-2149126 via The University of California – Davis.

M. Bergbauer was supported by the German Research Foundation (DFG) under the project "High-Performance Cut Discontinuous Galerkin Methods for Flow Problems and Surface-Coupled Multiphysics Problems" Grant Agreement No. 456365667.

B. Blais was supported by the National Science and Engineering Research Council of Canada (NSERC) through the RGPIN-2020-04510 Discovery Grant and the MMIAOW Canada Research Level 2 in Computer-Assisted Design and Scale-up of Alternative Energy Vectors for Sustainable Chemical Processes.

M. Fehling was partially supported by the ERC-CZ grant LL2105 CONTACT, funded by the Czech Ministry of Education, Youth and Sports. He was also partially supported by the Charles University Research Centre Program No. UNCE/24/SCI/005.

R. Gassmöller was also partially supported by NSF Awards EAR-1925677 and EAR-2054605.

T. Heister was also partially supported by NSF Awards OAC-2015848, EAR-1925575, and OAC-2410848.

L. Heltai is a member of Gruppo Nazionale per il Calcolo Scientifico (GNCS) of Istituto Nazionale di Alta Matematica (INdAM). LH was partially supported by the Italian Ministry of University and Research (MUR), under the grant MUR PRIN 2022 No. 2022WKWZA8 "Immersed methods for multiscale and multiphysics problems (IMMEDIATE)", and acknowledges the MIUR Excellence Department Project awarded to the Department of Mathematics, University of Pisa, CUP I57G22000700001.

M. Kronbichler was partially supported by the German Federal Ministry of Research, Technology and Space, project "PDExa: Optimized software methods for solving partial differential equations on exascale supercomputers", grant agreement No. 16ME0637K.

M. Kronbichler and L. Heltai were partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (call HORIZON-EUROHPC-JU-2023-COE-03, grant agreement No. 101172493 "dealii-X: an Exascale Framework for Digital Twins of the Human Body").

M. Maier was partially supported by NSF Award DMS-2045636 and by the Air Force Office of Scientific Research under grant/contract number FA9550-23-1-0007.

D. Wells was supported by NSF Award OAC-1931516.

Charles University is acknowledged for providing computing time on the Sněhurka cluster.

This research used in part resources on the Palmetto Cluster at Clemson University under National Science Foundation awards MRI 1228312, II NEW 1405767, MRI 1725573, and MRI 2018069. The views expressed in this article do not necessarily represent the views of NSF or the United States government.

References

[1] P. C. Africa, D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, S. Kinnewig, M. Kronbichler, M. Maier, P. Munch, M. Schreter-Fleischhacker, J. P. Thiele, B. Turcksin, D. Wells, and V. Yushutin. The deal.II library, version 9.6. *Journal of Numerical Mathematics*, 32(4):369–380, 2024.

[2] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software*, 45(1):2/1–26, 2019.

[3] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[5] H. Anzt, T. Cojean, Y.-C. Chen, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, and Y.-H. Tsai. Ginkgo: A high performance numerical linear algebra library. *Journal of Open Source Software*, 5(52):2260, 2020.

[6] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y. M. Tsai, and E. S. Quintana-Ortí. Ginkgo: A modern linear operator algebra framework for high performance computing. *ACM Transactions on Mathematical Software*, 48(1):2/1–33, 2022.

[7] D. Arndt, W. Bangerth, M. Bergbauer, M. Feder, M. Fehling, J. Heinz, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, B. Turcksin, D. Wells, and S. Zampini. The `deal.II` library, version 9.5. *Journal of Numerical Mathematics*, 31(3):231–246, 2023.

[8] D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Proell, K. Simon, B. Turcksin, D. Wells, and J. Zhang. The `deal.II` library, version 9.3. *Journal of Numerical Mathematics*, 29(3):171–186, 2021.

[9] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The deal.II finite element library: Design, features, and insights. *Computers & Mathematics with Applications*, 81:407–422, 2021.

[10] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith", H. Suh, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.23, Argonne National Laboratory, 2025.

[11] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc Web page. https://petsc.org/, 2025.

[12] W. Bangerth. Experience converting a large mathematical software package written in C++ to C++20 modules. *arXiv preprint* http://arxiv.org/abs/2506.21654, 2025.

[13] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software*, 38(2):14/1–28, 2012.

[14] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general purpose object oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24–es, 2007.

[15] W. Bangerth and O. Kayser-Herold. Data structures and requirements for *hp* finite element software. *ACM Transactions on Mathematical Software*, 36(1):4/1–31, 2009.

[16] M. Bergbauer, P. Munch, W. A. Wall, and M. Kronbichler. High-performance matrix-free unfitted finite element operator evaluation. *SIAM Journal on Scientific Computing*, 47(3):B665–B689, 2025.

[17] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[18] Boost C++ Libraries. http://www.boost.org/.

[19] W. E. Brown. A Proposal for the World's Dumbest Smart Pointer, v4, 2014.

[20] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

[21] T. C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler. A flexible, parallel, adaptive geometric multigrid method for FEM. *ACM Transactions on Mathematical Software*, 47(1):7/1–27, 2021.

[22] T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30:196–199, 2004.

[23] D. Davydov, T. Gerasimov, J.-P. Pelteret, and P. Steinmann. Convergence study of the *h*-adaptive PUM and the *hp*-adaptive FEM applied to eigenvalue problems in quantum mechanics. *Advanced Modeling and Simulation in Engineering Sciences*, 4(1):7, Dec 2017.

[24] A. DeSimone, L. Heltai, and C. Manigrasso. Tools for the Solution of PDEs Defined on Curved Manifolds with deal.II. Technical Report 42/2009/M, SISSA, 2009.

[25] M. Fehling and W. Bangerth. Algorithms for parallel generic *hp*-adaptive finite element software. *ACM Transactions on Mathematical Software*, 49(3):25/1–26, 2023.

[26] S. Filippone and M. Colajanni. PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Transactions on Mathematical Software*, 26(4):527–550, 2000.

[27] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Ltd., 3rd edition, 2009.

[28] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos. Enabling new flexibility in the sundials suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 48(3):1–24, 2022.

[29] R. Gassmöller, H. Lokavarapu, E. Heien, E. G. Puckett, and W. Bangerth. Flexible and scalable particle-in-cell methods with adaptive mesh refinement for geodynamic computations. *Geochemistry, Geophysics, Geosystems*, 19(9):3596–3604, 2018.

[30] C. Geuzaine and J.-F. 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.

[31] N. Giuliani, A. Mola, and L. Heltai. π-BEM: A flexible parallel implementation for adaptive, geometry aware, and high order boundary element methods. *Advances in Engineering Software*, 121:39–58, July 2018.

[32] C. Greif, T. Rees, and D. B. Szyld. GMRES with multiple preconditioners. *SeMA Journal*, 74(2):213–231, 2016.

[33] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[34] GSL: GNU Scientific Library. http://www.gnu.org/software/gsl.

[35] L. Heltai, W. Bangerth, M. Kronbichler, and A. Mola. Propagating geometry information to finite element computations. *ACM Transactions on Mathematical Software*, 47(4):32/1–30, 2021.

[36] L. Heltai and A. Mola. Towards the Integration of CAD and FEM using open source libraries: a Collection of deal.II Manifold Wrappers for the OpenCASCADE Library. Technical report, SISSA, 2015.

[37] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin. Taskflow: A lightweight parallel and heterogeneous task graph computing system. *IEEE Transactions on Parallel and Distributed Systems*, 33(6):1303–1320, 2021.

[38] B. Janssen and G. Kanschat. Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. *SIAM J. Sci. Comput.*, 33(4):2095–2114, 2011.

[39] A. Javeed, D. P. Kouri, D. Ridzal, and G. Von Winckel. Get ROL-ing: An introduction to Sandia's rapid optimization library. In *7th International Conference on Continuous Optimization*, 2022.

[40] G. Kanschat. Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. *Comput. & Struct.*, 82(28):2437–2445, 2004.

[41] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[42] M. Kronbichler and K. Kormann. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids*, 63:135–147, 2012.

[43] M. Kronbichler and K. Kormann. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM Transactions on Mathematical Software*, 45(3):29/1–40, 2019.

[44] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, Philadelphia, 1998.

[45] List of changes for `deal.II` release 9.7. https://dealii.org/developer/doxygen/deal.II/changes_between_9_6_0_and_9_7_0.html.

[46] Magic Enum C++. https://github.com/Neargye/magic_enum.

[47] M. Maier, M. Bardelloni, and L. Heltai. LinearOperator – a generic, high-level expression syntax for linear algebra. *Computers and Mathematics with Applications*, 72(1):1–24, 2016.

[48] M. Maier, M. Bardelloni, and L. Heltai. LinearOperator Benchmarks, Version 1.0.0, 2016.

[49] M. Mayr, A. Heinlein, C. Glusa, S. Rajamanickam, M. Arnst, R. Bartlett, L. Berger-Vergiat, E. Boman, K. Devine, G. Harper, et al. Trilinos: Enabling scientific computing across diverse hardware architectures at scale. *arXiv preprint arXiv:2503.08126*, 2025.

[50] P. Munch, T. Heister, L. Prieto Saavedra, and M. Kronbichler. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. *ACM Transactions on Parallel Computing*, 10(1):3/1–38, 2023.

[51] muparser: Fast Math Parser Library. https://beltoforion.de/en/muparser.

[52] OpenCASCADE: Open CASCADE Technology, 3D modeling & numerical simulation. http://www.opencascade.org/.

[53] W. Pazner, T. Kolev, and C. R. Dohrmann. Low-order preconditioning for the high-order finite element de Rham complex. *SIAM Journal on Scientific Computing*, 45(2):A675–A702, 2023.

[54] A. Prokopenko, D. Lebrun-Grandié, D. Arndt, and B. Turcksin. Arborx 2.0, 04 2025.

[55] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[56] J. E. Roman, F. Alvarruiz, C. Campos, L. Dalcin, P. Jolivet, and A. Lamas Daviña. Improvements to slepc in releases 3.14–3.18. *ACM Transactions on Mathematical Software*, 49(3):1–11, 2023.

[57] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[58] A. Sartori, N. Giuliani, M. Bardelloni, and L. Heltai. deal2lkit: A toolkit library for high performance programming in deal.II. *SoftwareX*, 7:318–327, 2018.

[59] T. Schulze, A. Gessler, K. Kulling, D. Nadlinger, J. Klein, M. Sibly, and M. Gubisch. Open asset import library (assimp). https://github.com/assimp/assimp, 2021.

[60] SymEngine: fast symbolic manipulation library, written in C++. https://symengine.org/.

[61] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024.

[62] The HDF Group. Hierarchical Data Format, version 5, 2025. http://www.hdfgroup.org/HDF5/.

[63] The Trilinos Project Team. *The Trilinos Project Website*. https://trilinos.github.io/.

[64] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, 2022.

[65] B. Turcksin, M. Kronbichler, and W. Bangerth. *WorkStream* – a design pattern for multicore-enabled finite element computations. *ACM Transactions on Mathematical Software*, 43(1):2/1–29, 2016.

[66] VTK. https://vtk.org.

[67] M. Wichrowski. Matrix-Free Ghost Penalty Evaluation via Tensor Product Factorization. *arXiv preprint arXiv:2503.00246*, 2025.

[68] M. Wichrowski, P. Munch, M. Kronbichler, and G. Kanschat. Smoothers with localized residual computations for geometric multigrid methods for higher-order finite elements. *SIAM Journal on Scientific Computing*, 47(3):B645–B664, May 2025.

[69] J. Zarestky, M. Bigler, M. Brazile, T. Lopes, and W. Bangerth. Reflective writing supports metacognition and self-regulation in graduate computational science and engineering. *Computers and Education Open*, 3:100085/1–12, 2022.

[70] zlib. https://zlib.net.