Roadmap for Software for UK Particle Physics

Towards a Collaborative Computational Project on Theoretical and Experimental Particle Physics

Executive summary

High-energy particle physics, both theory and experiment, aims to address some of STFC's key science challenge problems, including *What are the fundamental particles and fields?*, *What are the fundamental laws and symmetries of physics?* and *Why is there more matter than antimatter?* To that end, it has for many decades made heavy and effective use of high-performance and high-throughput computational resources. The lack of software engineering-specific funding for particle physics to date (beyond that specifically focussing on high performance), combined with the culture of the discipline, has meant that the software has grown relatively isolated from the wider software engineering and research software engineering communities. There are many areas where the software outperforms its peers in other disciplines, but also a number of areas that would benefit from attention.

In the document below, we outline the results of discussions held at workshops with members of the experiment and theory communities, and possible actions that may be supported by CCP-TEPP. Here we outline the key points, and refer to the full text for more detail.

A key challenge for the whole community is the push towards GPU resources. This will ultimately give significantly more computational power per unit cost, but is not without the need for an initial investment to achieve this. In some cases it requires new software development and algorithmic implementation; in others, it requires support in transitioning to make use of existing solutions.

At the same time, groups are starting to make use of more powerful algorithms to access physics that cannot be probed by existing implementations or to dramatically improve the efficiency of existing work—for example, the Hierarchically Deflated Conjugate Gradient and Logarithmic Linear Relaxation algorithms, and GPU and machine-learning based approaches to Monte Carlo.

Funders increasingly require retention and publication of data, metadata, and analysis workflows; this requires storage infrastructure that has not historically been easily accessible by the theory community—namely storage elements compatible with the International Lattice Data Grid protocols, and extensions to relevant metadata schemata to allow sharing of more classes of data. As the cost of entry becomes increasingly high, it is important that analysis of

HPC-generated data is reproducible, in particular for hero computations that may never be replicated.

The speed at which new research can be done benefits from having well-maintained software, with robust test suites, comprehensive, up-to-date user- and developer-facing documentation, and consistent code styles. Much software in the field falls short of one or more of these goals; direct support is likely to help clear the backlog of maintenance.

As there will always be a need for researchers to contribute to or lead the development of software in the field, it is essential that researchers gain the skills necessary to avoid repeating past mistakes; CCP-TEPP will aim to curate collections of existing training in this space, and support creation of new content where appropriate courses do not already exist. It will also facilitate a summer school to kickstart a generation of particle physics software specialists.

There is also a clear need for long-term software expertise in the field, to retain institutional knowledge and perform ongoing maintenance of software suites. Finding appropriate career paths for this, beyond a typical academic or short-term project-funded role, is important. A central RSE team is not a good fit for such a role, since RSEs in such teams must be able to be flexibly deployed to projects in many disciplines as needed, not be seen as a parking space for a particle physicist.

There is room for more collaboration between different research groups within the community. For example, multiple groups in lattice are interested in adopting the Hadrons software tool, with unfamiliarity being the main barrier to entry. In some cases, greater adoption of standards would reduce the barrier to collaboration further—rather than being tied to a single toolchain, transitioning freely between tools would become easier. There is also potential overlap between parts of the particle physics community and other computational disciplines, including computational fluid dynamics, condensed matter physics, tomographic imaging, and nuclear engineering where a greater dialogue could see synergies emerge.

Contents

Introduction and background

Collaborative Computational Projects are a set of initiatives supported by CoSeC, the Computational Science Centre for Research Communities, to support computational disciplines focused on one or more pieces of common research software or computational techniques. Historically, this support has been limited to research in the remit of specific UK Research Councils; recently, CoSeC launched a call for New Communities, with all UKRI research being in scope. Towards a Collaborative Computational Project for Theoretical and Experimental Particle Physics (CCP-TEPP) aims to identify the scope and requirements of a possible future CCP in the particle physics space.

Particle physics in the UK is heavily dependent on software and computational resources that underpin its research. Experimental particle physics requires software and high-throughput (and increasingly high-performance) computing support for most aspects of its work, covering simulation, collection and analysis of the vast data produced by experiments such as the Large Hadron Collider. In theoretical particle physics, lattice quantum field theory specifically requires significant amounts of high-performance computing resources, and largely uses open, community-developed tooling.

The aim of this document is to provide a roadmap for development of community software and associated activities in particle physics in the UK for the next five years. To ensure that the whole community is represented, a series of workshops was held in the spring of 2025.

- On 2025-04-29–30, a workshop was held at the University of Edinburgh focusing on the software practices and requirements of the computational theoretical physics community, focusing primarily on lattice quantum field theory.
- On 2025-06-02–03, a workshop was held at the University of Warwick focusing on the software practices and requirements of the experimental particle physics community.

Subsequently, additional meetings were held with groups who, owing to scheduling conflicts, were unable to be represented at the workshops above.

In the following sections, we describe a number of topics that were discussed at these workshops. Some of these were identified in advance as areas for discussion, while others arose organically. In each case we outline the technical context, discuss the current state of the art in software for particle physics, and identify actions that will positively impact our software and its applications. A subset of these actions will form the basis of the technical work to be targeted during the second year of the project.

Current context

Scientific case

The primary driver of computational resource usage in theoretical particle physics is lattice quantum field theory (LFT). This is currently the only known method allowing first-principles computations of quantities in strongly-coupled quantum field theories such as QCD, the theory describing the strong interaction. To be able to compute precisely what the Standard Model of particle physics predicts such that these predictions can be tested experimentally, significant amount of LFT computation are necessary. Areas of research focused on in the UK are flavour physics (e.g. hadronic decay amplitudes, muon $g-2$) and hadronic spectroscopy and structure (masses of particles and resonances, form factors, decay amplitudes). The former of these includes work from the HPQCD, RBC/UKQCD, and QCDSF/UKQCD collaborations, and researchers in Cambridge, Edinburgh, Southampton, Plymouth, and Liverpool; the latter from the HPQCD and HadSpec collaborations, with researchers in Glasgow, Edinburgh, Plymouth, Liverpool, and Cambridge. In addition to these precision tests, research also tries to better understand the QCD phase diagram, by looking at high temperatures and densities, of relevance to heavy-ion collisions and studies of the early universe; this includes work from the FASTSUM collaboration, and researchers in Swansea and Liverpool. Finally, lattice can also compute in field theories that have yet to be observed in nature. The Standard Model is known to be incomplete, and so a low-energy effective theory of some other theory that becomes visible at higher energy scales (less than or equal to the Planck scale, and potentially providing explanations for phenomena such as dark matter, the hierarchy problem, and quantum gravity). Being able to perform non-perturbative computations in theories that are candidates for extensions to the Standard Model allows both making concrete predictions that can be tested or falsified in experiment, and also better understanding the space of theories, guiding theorists and theoretical phenomenologists towards more promising candidates. This work includes the TELOS and LSD collaborations, and researchers in Liverpool, Edinburgh, Plymouth, and Swansea. Representatives from all institutions mentioned have fed into the preparation of this document, either in the Edinburgh workshop, or in follow-up discussions for those who had scheduling conflicts on the workshop dates.

Computation in experimental particle physics (EPP) is primarily driven by the requirement to record (or simulate), reconstruct, and analyse *events*, records of particle interactions, collected by a detector system. These interactions may come from artificial fixed-target or colliding beams, or from natural sources, and the analysis of these data provides the primary test of the current Standard Model of particle physics and searches for new particles or fields beyond it. The UK's research programme in EPP is active across a range of experiments and facilities aiming to answer these questions, with ~25 universities participating including some overlap with those working in the theoretical area. Searches for new particles and fields are undertaken by both the Large Hadron Collider (LHC) experiments at CERN and low background experiments searching for dark matter such as LZ and DarkSide. Precision tests of the Standard

Model are undertaken by both flavour (such as LHCb, Belle-2, NA62) and neutrino and dark matter (including DUNE, T2K/Hyper-K, LEGEND, LZ, DarkSide) physics experiments. The ATLAS and CMS experiments at the LHC also specifically target the Higgs boson as the origin of mass in the Standard Model and how this connects to the universe. Though this programme is broad, the common theme is a requirement to increase data collection rates, and thus the number of events, both to observe rare processes or new physics and to increase the precision with which the current Standard Model can be tested. It is this science-driven increase in data volume that drives EPP's computational requirement to increase the efficiency and accuracy with which this data can be processed in a timely manner.

Straddling the space between experiment and theory is a broad category that we will refer to as phenomenology; this includes some researchers that theorists might consider to be experimentalists, and others that experimentalists may consider to be theorists. Work in this space includes feeding into the event generation pipelines for experiment, and high-order perturbative analytical computations, for example of beta functions and scattering amplitudes.

Computation doing that science

There are a variety of software suites ("codes") that implement the techniques of lattice quantum field theory. Each implements a different subset of the various algorithms, formulations, and (relevant for BSM physics) gauge groups and fermion representations that are used in UK LFT research. The primary codes used are:

Code	Origin	Lead Developers	Languages	Platforms	Generation algorithms	Fermion formulations	Gauge groups	Fermion representations	Used by	Used for
Grid	Edinburgh	Peter Boyle (BNL)	C++17	CPU (vector optimisations) GPU (NVIDIA, AMD, Intel)	HMC	Wilson Wilson Clover Exponentiated Wilson Clover Unimproved Staggered Highly Improved Staggered Domain Wall Möbius Domain Wall	SU(N), Sp($2N$)	Fundamental Adjoint Two-index symmetric Two-index antisymmetric	TELOS RBC/UKQCD	Generation Measurement
HiRep	UK	Claudio Pica (SDU), Antonio Rago (SDU)	C (with C++/Perl code generator)	CPU (vector agnostic, compiler optimised) GPU (NVIDIA, AMD, Intel)	Heat bath HMC LLR heat bath (in fork, no GPU support) LLR HMC (in development)	Wilson Wilson Clover Exponentiated Wilson Clover	SU(N), Sp($2N$) [in fork, no GPU support] SO(N)	Fundamental Adjoint Two-index symmetric Two-index antisymmetric	TELOS	Generation Measurement
QUDA	USA	Kate Clark (NVIDIA)	C++	CPU (vector agnostic) GPU (NVIDIA, AMD, Intel)	N/A (library only)	Wilson Wilson Clover Unimproved Staggered ASQTAD Highly Improved Staggered Domain Wall Möbius Domain Wall	SU(3)	Fundamental	HadSpec	Library only
MILC	USA	Unmaintained	C, AVX assembly	CPU (vector optimisations) GPU available via QUDA	HMC	Wilson Wilson Clover Unimproved Staggered	SU(3)	Fundamental	HPQCD	Measurement

Code	Origin	Lead Developers	Languages	Platforms	Generation algorithms	Fermion formulations	Gauge groups	Fermion representations	Used by	Used for
Chroma	USA	Jefferson Lab	C++	CPU, GPU available via QUDA	Heat bath HMC	Highly Improved Staggered Wilson Wilson Clover Unimproved Staggered ASQTAD Highly Improved Staggered Domain Wall	SU(N)	Fundamental	HadSpec	Generation Measurement
open QCD	CERN	Martin Lüscher (CERN)	C	CPU (vector optimisations) GPU (under development, not public)	HMC	Wilson	SU(3)	Fundamental	FASTSUM	Generation
Hadrons	Edinburgh	Antonin Portelli (Edinburgh)	C++	As for Grid	N/A (measurement only)	As for Grid	As for Grid	As for Grid	TELOS RBC/UKQCD	Measurement

In addition to the named collaborations, MILC, Chroma, and openQCD are also used by other UK researchers and groups not forming formally named collaborations. Additionally, there are some proprietary codes that are worth mentioning:

- There is a body of PyTorch-based code used for configuration generation using machine learning techniques, used in Edinburgh, driving significant resource usage on US machines. This is considered to still be in a prototype stage and not ready for publication yet.
- There is a body of code and code generation tooling for computing contractions, developed and used by the HadSpec collaboration. There are differences of opinion around opening this code: some members want to ensure that the significant development time results in returns for the collaboration, and worry that open sourcing the code would allow others to catch up; additionally, there is a concern that collaborators would become overburdened by support requests.

The majority of lattice computations may be broken into three phases:

1. Generation of Monte Carlo ensembles of configurations of a gauge field
2. Computation of observables on these ensembles
3. Final statistical analysis of the resulting observables

For UK collaborations, the majority of work in flavour physics and hadron spectroscopy focus on steps 2–3, using gauge ensembles generated by other groups. On the other hand, UK work on the QCD phase diagram and BSM lattice physics both involve substantial generation of new ensembles, in addition to observable computation and analysis.

There is also work to apply quantum computing technologies to lattice; currently quantum computing is not sufficiently mature for there to be standard reusable software frameworks for lattice, instead, most work involves programming directly against vendor-provided libraries such as Qiskit (IBM) and PennyLane (Xanadu).

Outside of high-performance computing, theory has other software needs: much analytical analysis is done with Mathematica (proprietary, commercial) and FORM (open source), and there are several quite specialised codes employed in the enumeration and evaluation of high-order Feynman diagrams. FORM's maintainer has recently retired, creating the need for a new community member to take on the maintenance burden.

Once data leave HPC, they still require analysis. Workflows for this are typically bespoke to the individual user; however, they frequently use common components. Examples of these include pyerrors (developed by Fabian Josvig, Edinburgh), pyobs (Mattia Bruno, Milano-Bicocca), and gvar, lsqfit, and corrfitter (G. Peter Lapage, Cornell), the latter of which also suffers from the issue of maintainers retiring.

> ➢ Action: Identify new maintainers for FORM and gvar/lsqfit/corrfitter

The independent event-based data model in experimental particle physics has led to the general adoption of software frameworks that process events in a sequence of stages organised as a pipeline. These stages are generally organised as follows:

1. Data Collection or Simulation
    a. Collection: reads data streaming from the detector, filtering events of interest to be stored to disk (so-called “triggering”)
    b. Simulation:
        i. Event Generation: modelling of the fundamental interactions in an “event”, e.g. the collision of two protons in the LHC beam, outputting the particles that will be created by this interaction and their energy, momenta, and other kinematic properties.
        ii. Detector Simulation: modelling of the transport of the particles created by the event generation stage through the detector geometry, outputting the recorded signals for each event.
2. Data Processing
    a. Tracking/Reconstruction: the raw outputs (“hits”) from the detector are grouped into coherent “events”, each event a coherent record of an interaction. In detectors such as those at the LHC and some neutrino experiments, “hits” are first clustered into coherent “tracks”, each track a record of the path/kinematics of a particle through the detector. Tracks are subsequently collated into larger hierarchies, matching them back to common interaction points, reconstructing the overall topology of the event.
    b. Physics Analysis: the data from the Tracking/Reconstruction stage is used to extract, filter, plot, fit, and review relevant physical quantities for the topic being studied.

Each experiment generally has its own custom implementation for this overarching framework and for many subcomponents of each stage. This is not unexpected as exact details of the stages are specific to each experiment, and the analysis stages are deliberately implemented separately to allow rigorous cross-validation of results. The software packages (“codes”) maintained at a broad level by the experimental community are thus those that provide the foundational capabilities needed to implement the higher level per-experiment software stacks. These codes include:

Code	Languages	Platforms	Capability
EVTGEN	C++	CPU	Event Generation (B/D Meson Decays)
GENIE	C++	CPU	Event Generation (Neutrino interactions)
SNEWPY	Python	CPU	Event Generation (Supernova neutrinos)
HERWIG	C++	CPU	Event Generation (lepton-lepton, lepton-hadron and hadron-hadron collisions)

PYTHIA	C++	CPU	Event Generation (collisions between electrons, protons, photons and heavy nuclei)
SHERPA	C++	CPU, GPU in Chilli (phase space) and Pepper (tree level matrix elements)	Event Generation (lepton-lepton, lepton-photon, photon-photon, lepton-hadron and hadron-hadron collisions).
LHAPDF	C++	CPU, GPU	Parton distribution function calculations
HEPMC3	C++ plus Python bindings	CPU	MC Generator Event Record format
NUHEPMC3	C++	CPU	Neutrino MC Generator Event Record format
RIVET	C++ plus Python bindings	CPU	Event Generator validation
GEANT4	C++ (C++17)	CPU (GPU R&D in AdePT and Celeritas projects)	Detector Simulation
FLUKA	Fortran	CPU	Detector Simulation (NB: closed source)
OPTICKS	C++/CUDA	GPU (NVIDIA)	Optical Photon transport on GPUs for detector simulation
ACTS	C++	CPU (GPU R&D in traccc project)	Tracking/ Reconstruction in collider experiments
PANDORA	C++	CPU (GPU for AI/ML algorithms)	Tracking/Reconstruction in neutrino experiments and collider experiments
ROOT	C++ plus Python bindings	CPU (NVIDIA GPU support in some components)	Data Persistency, I/O and Physics Analysis
SCIKIT-HEP	Python	CPU, GPU	Physics Analysis Tools and Libraries
GAUDI	C++ plus Python bindings	CPU (NVIDIA GPU support for some use cases)	Event processing pipeline (ATLAS, LHCb, LZ experiments)

C++ and Python (either directly or via bindings to an underlying C++ implementation) remain the primary languages, with Fortran use in some simulation codes. CUDA is now widely used in both production and development code to accelerate some, but not all, hotspots, though production use is largely limited to experiment-specific codes. Some use is made of performance portability frameworks, primarily Alpaka, though Kokkos and SYCL are also being investigated. Increased raw data rates, primarily at the LHC, are also leading to closer

integration of the Tracking/Reconstruction stage with online Data Collection, with GPU use critical to keep pace with these rates, an example being LHCb's Allen application.

There are also a range of distributed compute/data services underlying these primary data processing packages, such as for data management, calibration/conditions data, and job control. Generally, these are highly experiment specific, though some tools have started to be shared, such as ATLAS's AMI metadata cataloguing application, the Rucio data management framework, and the DIRAC workflow management system.

The growth in AI/ML techniques has led to increased adoption of both generative and inferential methods across all stages of the processing chain, using industry standard packages such as Keras, PyTorch and Tensorflow, alongside specific systems such as FastML.

Where do we want to go?

A large current driver of software work in particle physics is the transition to GPUs, to allow supercomputers to deliver higher performance and better energy efficiency.

Particle physics through LFT computation was at the forefront of driving use of GPUs for non-graphics computation; however, there remain some groups and codes across theory and experiment that do not support GPUs. As HPC centres transition to majority- or all-GPU systems, it becomes increasingly important for software to support running performantly on GPU in order for collaborations to retain access to HPC resources.

Currently all "GPU" machines also have a CPU in each node to run the portions of software that are not GPU-enabled, so can in principle run CPU-only workloads. However, this is a highly inefficient way of using them, both in terms of power and in terms of computation delivered per unit of capital cost, and is unlikely to be well received in technical evaluations by computing centres, and so we choose to discard this mode of operation from consideration. This leaves two options for running workloads:

1. Since GPU codes frequently do not stress the CPU portion of a machine, and nodes on large systems are typically allocated to single users or projects at a time, projects with a mix of CPU and GPU software may be able to run CPU-only workloads on the unused CPU cores from their GPU jobs.
2. If this is not possible, then adapting the code to support GPUs, or switching to a code already having this support, is needed.

Where it is possible to switch to existing GPU implementations, this clearly requires less effort than writing a GPU implementation from scratch. However, this is still not without cost: tests must be performed to demonstrate that the new implementation gives compatible results with the previous one. This may take a substantial fraction of the time of a PhD studentship or postdoctoral role, and any resulting publication would be less impactful than new work. Lattice groups in this position include:

- HadSpec collaboration: their closed-source contraction code has GPU support, but UK users are unfamiliar, so time is needed to onboard and test; there is also concern over the vendor of GPUs supported, since the code is primarily tested with NVIDIA and AMD, but the Cambridge facility where HadSpec typically uses time is Intel GPU-based.
- HPQCD: the MILC version currently in use is built for CPU. In principle MILC supports GPU via QUDA, but this requires testing and verification. An alternative would be switching to using Grid/Hadrons, but the same constraints apply.

  ➢ Action: Support groups that are transitioning to GPU in performing comparison studies of their current implementation and new-to-them, existing GPU implementations.

Codes that are yet to have an available GPU implementation is most common in experiment due to the complexity of the problem (e.g. detector simulation) or the challenge of integrating GPU code into software stacks that are distributed to, and run on, the heterogeneous Worldwide LHC Computing Grid (WLCG). In Lattice, codes without a GPU implementation are typically investigating theories that receive less attention, so they do not benefit as directly from the work of other international collaborations. For example, NVIDIA targeted the most common SU(3) case when developing QUDA, meaning it can't be used for theories with other gauge groups.

- TELOS makes use of a fork of HiRep without GPU support, and a branch of this fork that adds support for the LLR algorithm. They have already discussed the pathway to merging their contributions back into the upstream repository with the maintainers; this requires some rewriting of the work to date to match the code standards for new contributions to the upstream repository. This would however give access to GPU support.

  - Action: Support TELOS in feeding the changes supporting Sp(2 N) theories into the HiRep upstream.

In programming GPU implementations, there is a common desire to move to (device) vendor-neutral APIs such as Alpaka, Kokkos and SYCL to avoid vendor lock-in and allow use of heterogeneous resources. The major questions here are which framework to choose, and reproducibility of results across different devices, even from the same vendor.

- Action: Explore options to support projects in testing different frameworks for reproducibility

While the shift to GPUs is a major driver for change in software, it is not the only one.

There is a shared desire to improve documentation, and maintainability of that documentation, at all levels, to improve the process of onboarding new users, collaborators, and RSEs into using and modifying the code. Similarly, improving testing and test coverage from unit to regression level was a common theme to increase robustness, reliability and assist developers in modifying the code (See also Software Engineering section below).

In addition to allowing easier implementation on GPUs, there are additional benefits to improving compatibility between different software suites. While the need to maintain multiple independent implementations to allow validation of results is broadly recognised (and so there is no appetite to attempt to converge on a single "universal code"), there are definite benefits to using common formats and tools for aspects that are less central to the physics analysis. In Lattice, one aspect is ensuring that tooling can work with the International Lattice Data Grid (ILDG) formats for data interchange, allowing better compliance with FAIR standards. Recent work by ILDG also introduced new more storage-efficient versions of the

formats, which are essential to ensuring that projects can make efficient use of their limited available storage. (See also Data section below.)

- Action: Implement ILDG read and write in HiRep
- Action: Implement reduced ILDG read and write in Grid

As experiments increase data rates and increasingly utilize AI/ML methods in each stage of the processing chain, it was noted that there is a growing requirement for a provenance tracking system that can record detailed metadata about how, for example, a particular result was arrived at. This could include aspects such as software version(s), CPU/GPU devices used, input parameters and datasets and so on, aiding reproduction or comparison against slightly different settings. A related need is to record validations/regressions of testing results for different versions of the same software over time, allowing changes and updates to be cross-checked against such processing provenances. Many such systems exist, but not as stable common implementations

- Action: review existing data tracking/provenance systems and data validation/plot systems for commonality to determine if a common tool/system would be possible.

As HPC systems get larger towards the exascale (and beyond, outside UK), individual lattice computations have not necessarily scaled at the same rate. The optimal way to utilise these machines becomes running large ensembles of jobs, either for parameter scans, or to obtain improved statistics. In some cases this can be done trivially using tooling built into the scheduler; however, in others, there are more complex dependencies or ways of sharing resources (for example, hybrid workloads with one job using GPU resources and another using CPUs of the same nodes), where standard schedulers such as Slurm do not provide the flexibility to encode the needed resources.

- Action: Identify and adopt, adapt, or implement tooling for managing ensemble workflows on HPC.

While all Lattice collaborations are looking to perform new work over the coming years, in some cases this is a new deployment of their existing tools, while in other cases new functionality is needed to enable the work to take place.

- Action: RBC/UKQCD: finalise and deploy implementation of HDCG in Grid/Hadrons
- Action: TELOS: implement LLR-HMC in Grid, without and with dynamical fermions

The (currently closed-source) software stack for machine learning-based Monte Carlo generation developed at Edinburgh currently does not scale to large node counts as well as more established lattice software suites; there is an ambition to improve this by integrating with Grid.

- Action: Integrate Grid and ML-based HMC software suite, delivering more efficient ML-based sampling methods to Grid users, and better parallel scalability to existing ML users.

What do we need to do that?

People and Skills

Experimental and theoretical particle physics have historically produced and attracted many talented software specialists, some of whom have stayed and become academics, but many of whom have left for other fields that better recognise software skills (and in many cases made significant impacts there).

There is however space for improvement in how such specialists are retained, and how their expertise is shared. Anecdotally, many researchers who would spend their time focusing on software excellence have felt pressured to either instead focus on physics at the expense of software, or to leave the field entirely. The Research Software Engineer role, first coined under that name in 2012, and the corresponding movement, was intended to provide a way to retain this expertise in academia, and this has been successful in that successive Research Software Engineer Surveys have shown that the plurality of RSEs in the UK have a background in Physics and Astronomy. However, many of these RSEs focus on topics outside of particle physics.

We consider two main models of employing RSEs. Firstly, there are embedded RSEs, who are attached to a specific research programme, either long-term in a similar role to staff scientists, or on short-term postdoc-style contracts. On the other hand, there are pool RSEs, who sit in central teams, and jump onto projects in many disciplines as and when they are needed. The latter group have been more successful at finding sustainable funding models and career pathways with progression opportunities.

The two models offer complementary benefits to the field. Pool RSEs can and do offer significant contributions in short interventions; the most common use case in lattice is for porting and optimisation work, from RSE teams that are specifically set up to enable such work. They bring a detailed knowledge of the target architectures, and experience of performing similar interventions on other software. There is always however a substantial set-up cost as the RSE becomes sufficiently familiar with the codebase to perform useful work.

There was broad recognition at both events that there is a need for long-term RSE attachments to research groups, with similar career progression pathways to academics. The benefits of having long-term embedded RSEs is retaining the in-depth knowledge of the software, as well as performing the regular ongoing maintenance (of software, documentation, testing infrastructure, etc.) that does not align with typical pool RSE project calls, but that postdocs and academics are not incentivised to keep on top of. They may act as a contact point between pool RSEs and the research community, allowing the pool RSE to be more effective more quickly, reducing the time requirement for onboarding and getting up to speed. Short-term embedded roles don't meet this aim to the same extent—a lot of effort is spent upskilling and integrating into the team, only to be back at square one 2-3 years later.

Long-term embedded RSE posts are rare in lattice currently: DiRAC supports one such role at the University of Edinburgh, and there is one STFC RSE Fellow in a similar role in Swansea (albeit focusing specifically on reproducibility of data analysis rather than HPC software). Some groups have applied for long-term RSE support (at the 10–20% of a person level) as part of STFC Consolidated Grant proposals; however, this will always be tensioned against requests for postdoctoral researchers. Experimental particle physics makes use of "core" positions within consolidated grant applications to retain this kind of technical expertise long-term; historically these have not been used by theory consolidated grants. Experimental consolidated grants also allow the "physicist programmer" role; however, this is most typically used for system administration support rather than research software engineers. Greater recognition of software and other non-paper research outputs (c.f. the Hidden REF project) will also support this; a return of dedicated long-term fellowships leading to permanent appointments would also be welcome.

There was discussion of the right balance of long-term embedded RSE support; it was agreed that the current level is not sufficient for our ambitions, but that 1 FTE per 3–4 FTE of academic staff may be enough.

> ➢ Action: Support long-term career and progression pathways for embedded RSEs.

It is however not possible for embedded and pool RSEs to take on the full burden of ensuring that software remains maintained—this would encourage other researchers to write unmaintainable code and throw it at RSEs demanding it be fixed. It's important that all new researchers (in particular, PhD students and postdocs) gain the skills necessary to perform the basics of code hygiene for themselves—for example, writing unit tests and documentation for new functionality that they write in pursuit of their research aims. The focus in general needs to be on maximizing maintainability and the ability of the software to be useful in future, rather than pursuing beautiful code for its own sake.

This is also where we anticipate the next generation of RSEs to emerge from. While historically RSEs have self-selected organically and pursued their software skills independently, having a training programme to give learners a grounding the basics and the pathway to develop further would be a force multiplier for those inclined to pursue that pathway, and would likely combat impostor syndrome in the process, where those who have learned independently feel (rightly or otherwise) underqualified for the work they are trying to do.

There is an option to embed some of these skills earlier in the curriculum—for example, as a replacement in the timetable for BSc projects for those doing an MPhys. However, since many new research students and postdocs come from an international background, adjusting the UK curricula in isolation will not fully address the need here.

This will always be in tension with the need to produce physics outcomes during a studentship or postdoc in support of applications for the next position; however, with the support of long-term embedded RSEs keeping the software maintainable, the benefits of this should be sufficiently clear to justify the (ultimately relatively small) effort required from the side of the researcher.

There is also a question of how much content can be taught centrally (within institutions or e.g. by HPC centres, to a multidisciplinary audience), and how much is more specific to the needs of particle physics.

- Action: Work to identify the basic skillset that new researchers need—e.g. C++, GitHub, version control, testing, CI, repository hygiene, etc.
- Action: Identify or develop, signpost, and deliver training on these skills for new researchers.
- Action: Formulate and deliver a summer school on some or all of these topics.

Many of these points have been made many times before in other documents; what remains unclear is how to fund this effort, and how to convince universities to create these career paths and roles.

Data infrastructure

The STFC and draft UKRI data policies require retention and sharing of data for an extended period (in the UKRI case 10 years following the most recent access to it). Both lattice and experiment produce sufficient volumes of data as to require dedicated infrastructure to retain and share data; general-purpose repositories are insufficient. The orders of magnitude are however quite different—the total volume of lattice field configurations is a single-digit number of petabytes, while for LHC data it in the exabyte range.

In fully computational workflows such as lattice and event generation/detector simulation, where software can be demonstrated to be fully reproducible, it may be treated as an extremely effective (but equally expensive) compression mechanism, requiring only storing the input parameters. However, for leading-edge research, recomputing data frequently requires orders of magnitude more resources than storing them. Many groups are unaware of or unable to comply with the long-term storage requirements for non-trivial (e.g. field configuration, propagators) data with current resources. While end-stage data (correlation functions, final fit results) may be shared using general-purpose platforms such as Zenodo or as supplementary material to papers, frequently, larger data are retained only on the HPC on which they are generated, and are lost as these machines go offline or purge data from expired projects. DiRAC has in recent years contracted with the cloud provider StorJ to make storage available to DiRAC projects, but whether this will comply long-term is unclear.

Lattice quantum field theory benefits from a well-developed set of metadata schemata and standards for sharing of field configuration data, from the International Lattice Data Grid (ILDG). The ILDG is however unable to provide storage directly, relying on provision of storage elements implementing standard transfer protocols. Where individual groups do have access to local storage infrastructure, there are challenges in making this available in a compatible way.

- Action: Support local groups in making local storage infrastructure available via ILDG.

In some collaborations, there remains a reticence among some collaboration members to share data that may form the basis of future work, even where these data are themselves built on open data.

There is also space for more robust solutions for cataloguing of data—the most frequently deployed technology in this space in lattice is the spreadsheet.

- Action: Support groups in identifying (or developing, if necessary) a cataloguing solution suitable for the needs of lattice theorists.

In experiment, where data arise from observation so cannot be recomputed, infrastructure for and awareness of the need for data retention are better developed. However, it was noted that knowledge about the availability and use of these resources is not well advertised or documented.

- Action: Given that the volume of lattice data is orders of magnitude smaller than experimental data, explore options for utilising GridPP infrastructure for lattice data storage as an ILDG storage element.
- Action: Support GridPP in advertising and documenting availability of and “how tos”on storage/compute resources.

Reproducibility

Adopting the definition that reproducibility is the ability of another researcher to take the data and analysis steps for a given publication, and repeat them obtaining the same result, we divide discussion of reproducibility into two categories: steps requiring large compute, that are anticipated to be done once (generating a gauge field ensemble, computing a correlation function, triggering and recording a given collider event), and “data analysis” steps that are anticipated to be repeated or potentially done in multiple ways, on laptops or modest local clusters (e.g. fitting correlation functions, generating plots and tabular data for publication).

Both categories of reproducibility are important: where steps are only performed once, it is important that there is confidence that they are correct. However, the significant diversity of

forms of data analysis means that it is valuable to readers to have access to a full workflow to fully understand the nature of the analysis that has been performed.

It was widely agreed that full bitwise reproducibility was neither achievable nor desirable. Since different platforms will give different results for the same basic operations, forcing bitwise reproducibility would significantly reduce the available computational power, since many common optimisations would be unavailable. (Bitwise reproducibility on the same hardware remains desirable, although modern hardware developments make this harder to achieve.) Reproducibility of results within uncertainties remains essential: without this, there is an obvious, uncontrolled and unaccounted uncertainty in results.

Reproducibility efforts in lattice have historically focused on use of large-scale compute; more recently there has been an effort within some collaborations to make all data analysis fully and openly reproducible.

All open-source lattice and event/detector simulation codes in use in the UK use seeded pseudorandom number generators with persistence of random state between successive jobs, and use parallel libraries or algorithms preserving reproducibility. Reproducibility of configuration generation or event/detector physics is typically tested as part of the development workflow.

Most lattice collaborations report having automated workflows for data analysis; however, the degree of automation varies. In some cases this is push-button end-to-end automation with all free parameters encoded as metadata; in others, individual analysis steps are automated, but there is an element of making choices on the fly and coordinating the order of steps by hand. Depending on the volume of data and fitting techniques used, fits vary in requirement from a few minutes on a laptop to 1–2 days of a CPU cluster node.

Some groups make use of Zenodo for publishing analysis code, while others keep this closed.

- Action: Support groups wishing to open their code (for all stages in the computational workflow—HPC, HTC, or desktop; simulation, processing, or analysis) in doing so.

In general, it is desirable that software and analysis workflows (and data supporting them) be FAIR (Findable, Accessible, Interoperable, and Reusable). Interoperability in particular requires the use of standardised data formats and metadata schema; while ILDG defines these for configuration data, there is a lack of schemata for later-stage data such as correlation functions and other per-configuration observables.

- Action: Support development of schemata for post-HPC data forming inputs to analysis workflows

Software engineering

Testing

It is widely agreed that it is important that research software in particle physics be true and accurate implementations of the underlying equations, and that it reliably gives the intended results for a given set of inputs. Good research software engineering practice recommends that this be verified by a set of automated tests, run automatically when new changes are introduced to the code. These tests are in addition to extensive manual tests, which all groups described performing as part of their development process. There are many good reasons for this, two of which are particularly relevant for particle physics research, and both of which ultimately relate to being able to test unfamiliar parts of the software.

Firstly, while testing by hand works for software with a singular focus, once this broadens it is easy for changes to one use case of the software to unwittingly affect others. Within a single collaboration it may be possible to manage this, but for community codes with many users, automated testing is the only way to prevent different groups treading on each others' toes. Secondly, as stated elsewhere, we increasingly need the support of pool Research Software Engineers unfamiliar with the specifics of both our software and our science; for them to be able to confidently make improvements in the sections of code they are tasked to work on, they need to be able to verify that the modified code still gives the correct result.

One measurement of the degree to which a piece of software is tested is the "test coverage": the fraction of lines of code that are run by at least one test in the test suite. While 100% coverage doesn't guarantee flawless execution, and very well-tested software may still have sub-100% coverage, the number gives a useful starting point for discussions around the state of testing in a piece of code.

It was widely acknowledged by participants from both theory and experiment that even for widely-deployed community-developed software, the state of automated testing and documentation are not where one would ideally want them to be, but that this was challenging without additional resources to support this. A full comparative study of the software engineering techniques applied in each software suite is beyond the scope of this document and the workshops leading to it; we can however summarise the approximate state of play.

All public/community HPC-facing lattice software discussed has a suite of unit and integration tests. However, in most cases the coverage of this is well below 100%, both in terms of lines of code and in terms of compile options tested. In most cases, the test suite is not run in continuous integration. In all cases, the test tooling is bespoke, rather than making use of an off-the-shelf testing framework. For proprietary code, tests are rarer. In some cases, the test suite does not currently pass, and it is not clear at what point in time it last did.

Procedures are similar in public/community/specific experiment software, with test suites (of partial coverage) run regularly and before merge or deployment. Larger packages, particularly those in simulation, have more intensive pre-release regression and validation tests. The impact is lower on experiment-specific software, since those performing the tests have a fuller awareness of the use cases of the code within the experiment.

- Action: Support and advocate groups in expanding and increasing robustness and coverage of test suites.
- Action: Provide training in use of unit testing, especially for numerical/stochastic algorithms, in compiled languages (e.g. GoogleTest for C++)

Continuous integration

At the simplest level, continuous integration is a setup that automatically runs a set of checks at a regular interval. This will typically include the test suite (see previous section); common times to run are when a change is pushed, when a pull or merge request is opened, and at a fixed interval (e.g. nightly or weekly). This avoids introducing bugs where contributors forget to run the test suite before submitting changes, and allows monitoring of things like performance and test coverage over time. Software that runs a robust set of tests in continuous integration, and rejects changes that reduce coverage or cause tests to fail, is typically considered to be more reliable than that that does not.

Some lattice and many experimental codes make use of continuous integration. For lattice, this is typically coupled to an HPC system: for Grid, DiRAC runs a TeamCity attached to (and running on identical nodes to) the Tursa service, which can submit jobs to the cluster where required For HiRep, a custom GitHub Actions runner (running in the University of Southern Denmark UCloud instance) submits jobs to the EuroHPC LUMI system in Finland. Both of these are limited to a single GPU architecture—Tursa to NVIDIA, and LUMI to AMD—so currently neither can be tested on all supported GPU vendors. Runners for experiment codes are more diverse, ranging from the basic provision in GitHub/GitLab to larger systems available at facilities such as CERN, though use of full HPC systems is not, in general, used. One key challenge in experiment has been provision of GPU runners to projects and this has often been ad-hoc and dependent on where the project is hosted.

A CI deployment that runs the full test suite is invaluable in letting developers and users alike know whether the test suite is currently expected to pass, and if it has been passing, then when it has been broken. It is typically insufficient to rely on users running the test suite themselves.

- Action: Support groups in enabling their test suites to run in CI, including setting up on DiRAC/Edinburgh TeamCity CI service.

- Action: Explore options to support groups in integrating additional GPU architectures to their CI.

Grid's CI also performs *continuous deployment* on Tursa for a subset of Grid use cases; users working with QCD may always have access to the latest build of Grid. The CI is used for continuous benchmarking of both the code and the Tursa facility itself, ensuring that there are no performance regressions in either. HiRep's CI is similarly used for performance regression testing of the code, in addition to running the unit test suite. Continuous deployment in experiment is only generally adopted by the full experimental software suites. Whilst there is a desire to increase continuous benchmarking in experimental codes, especially for simulation, significant setup and compute resource to cover larger validation jobs would be required.

Code formatting

It has been said that there for every N programmers, there are at least $N + 1$ opinions on code formatting. However, it is generally agreed that poorly-formatted code is a barrier to understanding and maintaining the software it forms. A conscientious group of skilled developers may be able to produce highly legible code, even if they each adopt different conventions in the work they originate. However, for most research software, where contributors are frequently a succession of inexperienced or self-taught developers, whose priority is to implement the functionality they require rather than thinking about readability, a lack of standardisation on formatting leads to code that is particularly hard to read.

This does not mean that PhD students need to spend time formatting their code—for most languages there is tooling that will perform this automatically, imposing a standard consistent style/standard to the entire project. (Examples include clang-tidy and clang-format for C++ code, and black and ruff for Python.) Whilst the choice of style will not be to the taste of every developer, it prevents unproductive discussions ("bike-shedding") about formatting, allowing code reviews to focus on the fundamental design and functionality aspects. (This document does not intend to advocate for any *specific* set of formatting conventions, only that projects adopt one of their choice, and makes effective use of staandard tooling to ensure that all authors can easily remain consistent with it.)

It is not currently common for lattice software to have codified style conventions, nor for any pre-commit checks to be performed. HiRep and QUDA define styles for clang-format, and HiRep makes use of pre-commit hooks (but only to enforce basic whitespace and merge conflict checks, not clang-format). Experiment codes have generally moved towards providing at least style guidelines and clang-format tooling, though enforcement and integration in CI workflows remains patchy.

- Action: Support interested groups in adopting code formatting and pre-commit/CI style checks

Documentation

Software documentation takes many forms: manuals, user guides, and tutorials, module- and function-level documentation for tooling like Doxygen, line-level comments, and even variable and function names may be considered documentation. Different forms of documentation address different groups: introductory tutorials will target new users, while line-level comments are more likely to be useful to those making deep changes to the software. There are many benefits to having high-quality documentation at each level: new users are able to start their research more quickly and with less time required providing guidance; early results are more likely to be correct first time, with fewer instances of incorrect or nonsense results due to misconfiguring the software in non-obvious ways; and new developers (including both those who will be contributors long term, and pool RSEs making rapid targeted changes) are able to get up to speed more quickly and deliver greater results in a shorter time. Since time is an increasingly precious asset in academia, aiming for high-quality documentation is a natural target to ultimately enable more time to be spent on science.

Most software in particle physics has some form of documentation; most frequently, a manual, kept in the same repository as the code, but updated asynchronously. Thus, this manual is out of sync with the code and omits much information that would be useful to potential new users and developers. There is typically no direct link between the code and the manual, so when making code changes, authors are not alerted that they should update the relevant portion of the manual. Frequently, both are required to read the code to understand its full capabilities and the details of their implementation. However, the low quality of inline documentation within code can make this challenging. This forms a barrier to adoption of new tools—one must dive deep into the code to understand how to use the tool; this may drive users to write their own tools rather than use those that already exist, either because they are unaware of them, or because they better understand code they write themselves rather than undocumented code from others.

An example that was raised is that MILC depends heavily on a stack of USQCD libraries (QMP, QIO, QDP, etc.), and depends heavily on specific versions of these, but the compatible versions are undocumented, so extensive testing is needed on each version upgrade to ensure that the results obtained are compatible with those obtained from the previous version.

Some tools make use of automated toolkits such as Doxygen to generate documentation from the code; however, in most cases the additional markup necessary to take the output from skeleton to rich, readable output is not present.

A very common pattern in particle physics software development (albeit one inferred from the resulting code rather than observed directly) is to begin by formulating or reading a system of equations (with functions and symbols defined with symbols like D, x, i, μ), and then translate this symbol-for-symbol into functions and variables in the code. Descriptive names can make code much more approachable by non-specialists (even if the terminology is not

familiar, it is easier to formulate useful search queries with multiple words than with single letters); however, this is not essential. What is essential is having a reference to a paper or textbook defining the symbols and methodologies being used, so a new developer may refer to them to understand the intended meaning of the symbols—since different groups frequently have different notation conventions. This is, unfortunately, missing in the majority of particle physics software.

Frequently, updating documentation is viewed as low priority compared to other activities (including searching for future jobs, as the work is typically done on short-term contracts), resulting in work being left undone. The time saved on understanding the code in the first place would likely outweigh the time spent updating documentation, but the former price is typically paid by someone other than the person avoiding spending the time on the necessary changes.

It may be possible to use LLM-based toolchains to accelerate the process of preparing documentation, particularly the boilerplate aspects. Expert input is however essential in providing the physics insight that such documentation must provide; pure LLM output is likely to invent false meanings for functions and variable names, and allude to functionality not present, which is less helpful than no documentation at all.

- Action: Identify or formulate appropriate documentation guidelines for computational particle physics software
- Action: Support groups in uplifting their software to meet these guidelines (e.g. via docathons).
- Action: Explore options to support groups in improving documentation and its maintainability.

When preparing documentation guidelines and documentation, it's important to consider the range of audiences who may need documentation and ensure that all their needs are covered. (Examples might include introductions for new users, for new feature developers, and for pool RSEs jumping into the code to perform porting or optimisation work, run-time optimisation guides for advanced users, and reference material for advanced users and experienced developers.)

There is also a degree to which it is necessary to train new developers (including PhD students and postdocs) on the etiquette and best practices of contributing to open-source projects in general. (Examples include updating documentation, conforming to existing code styles—both documented and undocumented, and formulating helpful commit messages), Many practices that seem obvious to experienced practitioners on reflection are not trivial for new contributors.

Some experiments have integrated C++/Python linting tools into their CI, though these can only check general conventions and adoption is not universal across the field.

- Action: Identify or develop, and deliver or promote, training around etiquette and best practice for contributing to software projects.

Discovery and keeping manuals, guides and tutorials up to date was a problem particularly noted by experiment due to the multi-decadal timescales they may operate over, with enforcement difficult as new features/performance are prioritised over long term support.

- Action: Seek out guidance on best practices in software documentation for very long-running software projects, in collaboration with CoSeC and others.

Build systems

A common rite of passage for new PhD students in physics software in general is to be able to compile the toolchain they are to use for the first time. This can be a challenging task even for experienced users: some tools have very long compile times or require very precise configuration directives to achieve the desired result. Some software packages include build scripts, such as Makefiles, either generic or machine-specific. However, the quality and reliability of these scripts can be variable, and in some cases can fall out of sync with development and/or compute platforms, giving sub-optimal or failing builds and increasing the barrier to understanding how to compile the code from scratch. Long compile times specifically can be a barrier to development of software. If each code change results in 10, 30, or even 120 minutes of compile time before the change can be tested, this breaks the standard modify–build–test cycle of software development, significantly reducing the pace at which work can be performed.

Ensuring that the required library dependencies for the software are available on the target compute platform (HPC or local machine) is frequently also a barrier as these may also suffer from the same issues. A more robust solution here is defining recipes for package managers such as Spack, which has been done, for example, for the Grid and Geant4 codes, to give examples from lattice and experiment.

- Action: Identify or formulate appropriate guidelines for build systems for computational particle physics software
- Action: Support groups in adopting better build systems and processes for their software
- Action: Improve build processes, both by providing up-to-date Spack package recipes for lattice and experiment tools, and by reducing the need for user intervention in the manual build process

The above discussions primarily focus on HPC-targeted code. Software for data analysis, that runs locally, is frequently viewed as write-only code. Some packages are retained (examples

include gvar, lsqfit, corrfitter, pyobs, pyerrors); however, the code that calls these typically do not have tests nor make use of CI. One collaboration (TELOS) is exploring the use of CI to run data analysis; this is still in early stages.

> ➢ Action: Support groups wishing to make their analysis workflows more robust, automated, or open

Hardware and Infrastructure

The majority of compute usage in lattice is from HPC-suitable batch workflows. For these, DiRAC provides CPU and GPU computing resources which are well used across the community. In addition, most groups source access to additional compute, either because of a shortage of capacity, or to allow greater flexibility (for example, in application processes, or in architectures available). Examples of other resources utilized include institutional and regional clusters such as Supercomputing Wales, EuroHPC facilities such as LUMI, and US facilities such as Aurora. Experimental use of compute is primarily through the GridPP and WLCG high-throughput systems, plus some use of institute/facility level clusters for development work. However, the growing level of interest in the use of AI/ML and GPU and national level growth in heterogenous HPC systems is leading to a realization of the need to utiltize these resources, but use remains limited.

The DiRAC hardware sites are ageing, and will at some point require replacement; currently, there is no sign as to if or when funding will be available for this. There have been discussions around a possible federated, pan-UKRI approach to the next generation of UK research HPC resources, but there is currently little clarity on future funding or what the governance of such a service may look like.

There were discussions as to gaining access to the AI Research Resource (AIRR) machine—Isambard-AI in Bristol and DAWN in Cambridge. For standard importance-sampling workflows the current setup appears to work better; however, as generative models become more integrated with Monte Carlo generation, there may be a benefit to see from this.

Also discussed was the ongoing transition to GPU compute, and the decreasing availability of CPU-only resources. While most lattice problems are well suited to GPU, not all are, and not all that are have been ported yet. Some experiment workflows, especially Monte Carlo, are challenging to adapt to GPU. For the time being, EuroHPC continue to offer large CPU resources, which are less oversubscribed than the diminishing number of UK ones.

> ➢ Action: Support groups in porting software to GPU, or migrating to solutions already supporting GPU.

As discussed in the data section above, a missing component in the hardware ecosystem supporting lattice is a long term data retention/preservation/curation resource. An example

that was raised from the US is the tiered storage system at Jefferson Laboratory, having 1PB of disk and a reportedly unlimited amount of tape. In some cases, the unavailability of storage (both on-system and for long-term storage) and the difficulty of data curation is limiting the ambition of the research.

A pain point of HPC users common between subsets of the lattice community and also the amplitudes community is the unsuitability of the queues provided to the workloads to be run. Certain workloads are relatively trivial to write and run if given sufficient time, but highly non-trivial to checkpoint; others rely on steps that must be serialized, meaning that needing to requeue at the end of a job significantly reduces the throughput of work through the machine. Similarly, experimental workflows generally assume a high throughput system and require adaptions at the workflow and code levels to adapt to heterogenous systems.

The section of the community making use of quantum computing technologies is currently able to meet its needs from a mix of public and private-sector hardware. General-purpose machines are sufficient for these group's needs; having co-designed machines as part of the next DiRAC iteration isn't seen as necessary, although it may be for a subsequent one.

Building bridges

One of the goals of CCP-TEPP is to enable greater collaboration and sharing of expertise between different parts of the particle physics community on topics relating to software. We should also look for synergies with other disciplines: what other groups may have similar challenges to particle physics, are there solutions that we can learn from them, and if not, could we collaborate on finding one.

Intradisciplinary collaboration

A view common to participants from theory and experiment is the importance of having multiple implementations. In experiment, having separate experiments (e.g. ATLAS and CMS) studying the same quantities, and performing independent analyses using independent toolchains, gives confidence in the robustness of the result. Similarly, in theory, having multiple independent implementations of the algorithms to be used maximises the opportunities to find errors in one implementation.

That said, there are still areas where greater commonality and sharing can be useful. No one lattice toolchain implements every single algorithm, and for exploratory studies it may be convenient to integrate tools from multiple different stacks into a single workflow. This requires interoperability between the tools, which is currently not at the level it needs to be to enable this.

> ➢ Action: Support work increasing interoperability of data between different lattice toolchains, particularly the implementation of the full ILDG binary file format v1.2.

As discussed above, there is also significant inertia to being able to switch toolchains, due to the effort needed to first understand the capabilities of the tool, then how to make use of these in an equivalent way to an existing workflow, and finally to validate that the deployment is correct and gives equivalent results to previous work. Expertise being siloed within institutions or collaborations can be a barrier to leveraging existing knowledge of the target tooling in the community, in cases where there would be the goodwill to support such efforts. Postdoctoral researchers moving between institutions is one way that knowledge propagates between sites, but this is relatively slow and results are not guaranteed. A lattice-specialist RSE team sitting centrally (e.g. within CoSeC, or at least working as a distributed team similarly to the DiRAC RSE team) may be able to overcome some of these barriers.

> ➢ Action: Support groups looking to switch tools in setup and validation of new tooling.

Tighter integration of different tools—e.g. coupling of different tools together at compile or run time to access features from both—is less likely to be successful. This is because the

libraries frequently use different memory layouts, and the features required sit within relatively tight loops, so the time to pass data across the interface would outweigh the time spent in useful computation. Redesigning tooling in a more composable way would be an entirely new software project, and would be significant effort with limited perceived gains; it would also have significant implications on performance. Tools like Grid and QEX come closes to this model, Grid at compile time and QEX at run time.

Cross- and inter-disciplinary collaboration

The majority of collaboration to date happens where there is a common scientific interest. Lattice results appear in the Particle Data Book, and have given inputs into the measurement of the muon anomalous magnetic moment; however, this collaboration is indirect—quoting published results rather than working together. In some cases, there may be direct interaction for one group to help the other interpret or otherwise make use of the former's published results. Sufficiently complex analysis outputs require significant hand-holding to get a potential user of the data sufficiently familiar to be able to work with them;

> ➢ Action: Support the development of cross-disciplinary schemata and/or ontologies to enable easier sharing of data across the community.

Phenomenologists in particular frequently sit near or straddle the interface, bridging gaps in understanding or communication. For those further from the interface, feedback from participants was that the algorithms and techniques being used are too dissimilar for sharing of code between, for example, lattice and experiment. As discussed above, however, the need for software engineering skills is similar across the board, giving scope for shared training.

> ➢ Action: Support cross-disciplinary training in software skills in areas where there is common need.

One avenue as yet less explored is the possibility to collaborate on determination of parton distribution functions, which is of interest to both theory and experiment.

Additionally, there is scope for collaboration at an infrastructural level—as discussed above, it may not make sense for LFT to maintain separate data infrastructure to the WLCG's, and as experiment begins to need more HPC resource, sharing with the theory community may be more efficient than buying and deploying separate infrastructure.

Lattice field theory techniques originally arose from studies of lattices in condensed matter systems; there remains some potential overlap there. In particular, while previously lattice quantum field theory was dominated by path integral approaches, Hamiltonian methods are increasingly common to both fields with the advent of quantum computing approaches; there may be some possibility for collaborative projects in that space. A barrier to this is a lack of

awareness within the particle physics community as to what is going on the condensed matter. There may also be commonalities with cold atom experiments.

> ➢ Action: With CoSeC support, facilitate greater knowledge sharing between lattice quantum field theory and condensed matter.

Additionally, CoSeC have highlighted in conversations that many of the techniques required in lattice computations (e.g. large sparse matrices, curve fitting) are areas where the CoSeC computational mathematics theme has developed significant expertise supporting other communities.

> ➢ Action: Build better links and enable greater knowledge sharing between the CoSeC computational mathematics theme and the lattice community.

Other areas touched on but not in detail include:

- Computational fluid dynamics: Structured grid approaches are not dissimilar from the hypercubic lattices used in LFT
- Tomographic imaging: Spectral reconstruction techniques may be similar to those being increasingly deployed in lattice
- Nuclear engineering: Monte Carlo techniques for particle transport are very similar, especially for neutral particles, and electrons/positrons/gammas.
- Astroparticle/Astrophysics: Supernova modelling/early warning systems in relation to neutrino experiments.

Barriers to more interdisciplinary collaboration include that different communities use different sets of jargon, so may be discussing the same thing with different vocabulary (so not realise it), or may use the same words to mean very different things (so talk past each other). Frequently this can mean that it isn't sensible for software to be combined; however, there are examples of this being done successfully—for example, where cosmology has overlap with computational fluid dynamics, there has been successful collaborative work on applying the tools and techniques from the latter in the former, once these commonalities were identified. Removing barriers remains expensive both in terms of time and goodwill, so should only be pursued where there are clear tangible benefits, rather than where the barriers are themselves beneficial to the communities in question.

Summary of actions

- Identify new maintainers for FORM and gvar/lsqfit/corrfitter
- Support groups that are transitioning to GPU in performing comparison studies of their current implementation and new-to-them, existing GPU implementations.
- Support TELOS in feeding the changes supporting Sp($2N$) theories into the HiRep upstream.
- Explore options to support projects in testing different frameworks for reproducibility
- Explore options to support groups in improving documentation and its maintainability.
- Improve build processes, both by providing up-to-date Spack package recipes for lattice tools, and by reducing the need for user intervention in the manual build process
- Provide training in use of unit testing for numerical/stochastic algorithms in compiled languages (e.g. GoogleTest for C++)
- Implement ILDG read and write in HiRep
- Implement reduced ILDG read and write in Grid
- Review existing data tracking/provenance systems and data validation/plot systems for commonality to determine if a common tool/system would be possible.
- Identify and adopt, adapt, or implement tooling for managing ensemble workflows on HPC.
- RBC/UKQCD: finalise and deploy implementation of HDCG in Grid/Hadrons
- TELOS: implement LLR-HMC in Grid, without and with dynamical fermions
- Integrate Grid and ML-based HMC software suite, delivering more efficient ML-based sampling methods to Grid users, and better parallel scalability to existing ML users.
- Support local groups in making local storage infrastructure available via ILDG.
- Support groups in identifying (or developing, if necessary) a cataloguing solution suitable for the needs of lattice theorists.
- Given that the volume of lattice data is orders of magnitude smaller than experimental data, explore options for utilising GridPP infrastructure for lattice data storage as an ILDG storage element.

- Action: Support groups wishing to open their code (for all stages in the computational workflow—HPC, HTC, or desktop; simulation, processing, or analysis) in doing so.
- Support development of schemata for post-HPC data forming inputs to analysis workflows
- Support and advocate groups in expanding and increasing robustness of test suites.
- Support groups in enabling their test suites to run in CI, including setting up on DiRAC/Edinburgh TeamCity CI service.
- Explore options to support groups in integrating additional GPU architectures to their CI.
- Support interested groups in adopting code formatting and pre-commit/CI style checks
- Identify or formulate appropriate documentation guidelines for computational particle physics software
- Support groups in uplifting their software to meet these guidelines (e.g. via docathons).
- Identify or develop, and deliver or promote, training around etiquette and best practice for contributing to software projects.
- Identify or formulate appropriate guidelines for build systems for computational particle physics software
- Support groups in adopting better build systems and processes for their software
- Support groups wishing to make their analysis workflows more robust, automated, or open
- Support long-term career and progression pathways for embedded RSEs.
- Work to identify the basic skillset that new researchers need—e.g. C++, GitHub, version control, testing, CI, repository hygiene, etc.
- Identify or develop, signpost, and deliver training on these skills for new researchers.
- Formulate and deliver a summer school on some or all of these topics.
- Support groups in porting software to GPU, or migrating to solutions already supporting GPU.
- Support work increasing interoperability of data between different lattice toolchains, in particular the implementation of the full ILDG binary file format v1.2.
- Support groups looking to switch tools in setup and validation of new tooling.
- Support the development of cross-disciplinary schemata and/or ontologies to enable easier sharing of data across the community.

- Support cross-disciplinary training in software skills in areas where there is common need.
- With CoSeC support, facilitate greater knowledge sharing between lattice quantum field theory and condensed matter.
- Action: Build better links and enable greater knowledge sharing between the CoSeC computational mathematics theme and the lattice community.