PEPPER: A Portable Parton-Level Event Generator

Enrico Bothmann[1], Taylor Childers[2], Walter Giele[3], Stefan Höche[3], Joshua Isaacson[3], and Max Knobbe[1]

[1]Institut für Theoretische Physik, Universität Göttingen, 37077 Göttingen, Germany
[2]Argonne National Laboratory, Lemont, IL, 60439, USA
[3]Fermi National Accelerator Laboratory, Batavia, IL, 60510, USA

E-mail: enrico.bothmann@uni-goettingen.de

Abstract. Parton-level event generators are one of the most computationally demanding parts of the simulation chain for the Large Hadron Collider. The rapid deployment of computing hardware different from the traditional CPU+RAM model in data centers around the world mandates a change in event generator design. These changes are required in order to provide economically and ecologically sustainable simulations for the high-luminosity era of the LHC. We present a complete leading-order parton-level event generation framework capable of utilizing most modern hardware, and discuss its performance in standard-candle processes at the LHC.

1 Introduction

The upcoming high-luminosity era at the LHC (HL-LHC) will provide unprecedented collision data statistics. Monte-Carlo event generators (MCEG) provide simulated collision event samples that can be directly compared to the experimental data samples, e.g. to predict signals, subtract backgrounds, calibrate detectors, or extract model parameters. Due to the expected excellent experimental precision, poor MCEG computing performance can limit experimental success of the HL-LHC [1, 2].

Most of the computing time spent on MCEG by the ATLAS and CMS collaborations comes from the regular production of large $pp \to V +$ jets and $pp \to t\bar{t} +$ jets event samples [3]. We can dub them "heavy-hitter" processes due to their dominant computational footprint. With their relatively large cross sections, even when requiring a number of extra jets, and with their generic final-state signature, a large number of physics analyses rely on their statistical precision in the respective relevant phase-space regions.

Where more detailed performance studies for standard simulation setups of those heavy-hitter processes have been done, we can even nail down the components of the MCEG simulations that most contribute to the computational footprint. For SHERPA [4], we know that more than two thirds of the computing time is spent on the evaluation of squared tree-level matrix elements, while squared loop matrix elements contribute at most about 10 % [5]. Everything else, such as PDF evaluation, parton showering or hadronization only contribute a few percent each and are at this point irrelevant as targets for further performance improvements.

Tree-level matrix elements contribute most in these setups, because the standard multijet merged samples include matrix elements for up to $n = 5$ ($n = 4$) additional jets for $pp \to V +$jets ($pp \to t\bar{t} +$jets), and the highest-multiplicity matrix elements are only evaluated at leading order, since next-to-leading-order (NLO) matrix elements would be prohibitively expensive. Tree-level matrix element algorithms scale at best with $\mathcal{O}(n^3)$ [6], and the unweighting efficiency of these high multiplicities are very low (one may need to calculate tens or hundreds of thousands of matrix elements before accepting an event for these multiplicities [7]). The two thirds of the overall computing time mentioned above is therefore dominated by the one or two highest-multiplicity tree-level matrix elements and their phase-space generation. Improving the performance of this component can therefore reduce the overall computing budget for these

processes by up to a factor of three. The proxy figure of merit for these improvements would be the unweighted event generation throughput for $pp \to e^+e^- + 5\,\text{jets}$ and $pp \to t\bar{t} + 4\,\text{jets}$.

In this contribution, we present the portable standalone tree-level matrix-element generator PEPPER, which can be compiled to use GPU accelerators achieving much higher event throughputs compared to using the CPU only, and allowing us to fully utilize the current and upcoming generation of exascale high-performance computing (HPC) resources.

Details on the first production-ready release of PEPPER are given in [8]. Other efforts to port parts of the MCEG toolchain are the MADGRAPH5_AMC@NLO on GPU project [9], the MADFLOW project [10], and the GAPS parton-shower [11]. An outline of a future common benchmarking of PEPPER and MADGRAPH5_AMC@NLO on GPU has been given in [12, Chap. II, Sec. 4].

2 Methods

PEPPER can be compiled for single-threaded CPU execution, for multi-threaded CPU execution, and for execution on a GPU device. This is achieved with a single codebase and using the KOKKOS C++ Performance Portability framework [13, 14] (we also maintain a CUDA version for performance comparison studies). Below, we will often discuss the parallelisation of different event generation steps; this of course only applies when the code has been compiled for multiple threads, otherwise the algorithms run serially on a single thread.

In PEPPER, each computing thread generates one event. Thus a batch of events is generated in parallel if more than one thread is available. To ensure data locality (and coalescent GPU memory reads/writes [15]), the event batch data is implemented as a structure of arrays, i.e. each event property (e.g. the event weight, or the energy component of the momentum of the first outgoing particle) is stored contiguously in memory for all events in the batch.

The first step in event generation is to generate the required random numbers for the Monte-Carlo integration. In PEPPER, all random numbers required for a single event batch are generated in parallel, before proceeding with the rest of the event generation pipeline.

The next step is to generate a phase-space point for each event in the batch. We use a simple phase-space algorithm, CHILI [16], which parametrizes phase-space using a single t-channel and an adjustable number of s-channel resonances. The internal implementation supports up to one s-channel (e.g. for the Z resonance in Drell-Yan production), and is executed in parallel; we call this the "Basic CHILI". The random number mapping is optimized using VEGAS [17].

The simulation then proceeds to evaluate the parton density functions (for hadronic collisions) and the strong coupling. This is done via a modified version of the LHAPDF library [18], which allows multi-threaded execution using KOKKOS and/or CUDA.

The squared matrix elements at tree level are then evaluated in parallel using Berends-Giele recursion relations [19] to provide a good scaling behavior with the final-state multiplicity. We combine helicity sampling and colour summing with a minimal colour basis to achieve optimal GPU performance [20, 21].

At this point of the simulation, we optionally apply a hit-or-miss algorithm in parallel if the user has requested an unweighted event sample. Given the typically low efficiencies at the relevant jet multiplicities, only a small number of events is accepted. When event generation is done on a GPU with its own RAM, this has the additional benefit that we only need to copy the data for these non-zero events to the host CPU, such that the copy time becomes relatively small compared to the evaluation time of the matrix elements. Since all event generation steps are done in parallel, we only need to copy data between the CPU and the GPU once, at the end of the event batch generation pipeline.

The events can then be written to disk using the HEPMC3 [22], LHEF3 [23] LHEH5 [7, 24] output formats. The latter is based on the HDF5 [25] database library which allows for efficient writing of large event files and which features collective MPI writing for a good scaling behaviour on large HPC clusters. LHEF3 and LHEH5 event files can be read in by SHERPA and PYTHIA [26, 27] for additional particle-level simulation steps, thus integrating PEPPER into the full event simulation toolchain. The LHEH5 files can even be used as input samples in multi-jet merged simulations in SHERPA [24].

3 Results

Before highlighting results of PEPPER, we plot results given in [16] for the unweighting efficiency achievable with the CHILI phase-space generator combined with VEGAS optimization. Figure 1 compares the full CHILI generator, labeled "Chili", with one that uses the minimal number of s-channels (1 for W^+ and Z production, 0 otherwise), labeled "Basic", with the efficiency of the recursive phase-space generator implemented in SHERPA [28] for various benchmark LHC processes. The obtained CHILI efficiencies are similar to the SHERPA efficiency, or higher by up to an order of magnitude, with the exceptions of $\gamma + n\,\text{jets}$ production and for $W^+ + 5\,\text{jets}$ production. See [16] for additional details.

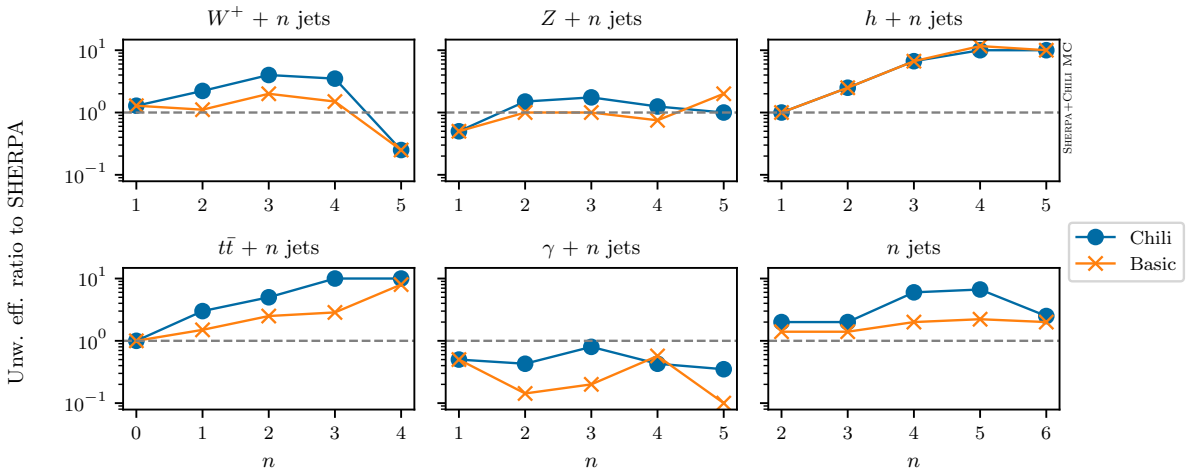

Figure 1: Results for the unweighting efficiency with the CHILI phase-space generator in its normal and "Basic" mode, comparing to the recursive phase-space generator of SHERPA. Data taken from [16].

MEvents / hour	2×Skylake8180	V100	A100	H100	MI100	MI250	PVC
$pp \to t\bar{t} + 4j$	0.06	0.5	1.0	1.7	0.4	0.3	0.3
$pp \to e^- e^+ + 5j$	0.003	0.03	0.05	0.1	0.03	0.03	0.02

Table 1: Parallel unweighted event generation rates of PEPPER, comparing multi-threaded CPU evaluation (2×Skylake8180, using all 56 threads) with GPU accelerated evaluation on various data-center GPU by Nvidia (V100, A100, H100), AMD (MI100, MI250) and Intel (PVC). Results taken from [8].

In Fig. 2, we show the single-threaded CPU performance for the unweighted event generation throughput of PEPPER for $pp \to e^+e^- + n\,\text{jets}$ and $pp \to t\bar{t} + n\,\text{jets}$, as first published with all details on the setup in [8]. This single-threaded throughput gives the baseline performance of PEPPER, without any parallel execution. The throughput is compared with COMIX [28], both using its recursive phase-space generator, and using an implementation of CHILI in SHERPA. The throughput of PEPPER is on par or higher, by up to an order of magnitude.

Finally, Tab. 1 gives the results for the unweighted event generation throughput for the highest jet multiplicities with parallel evaluation on various CPU (Intel Skylake8180) and GPU architectures (Nvidia V100, A100 and H100; AMD MI100 and MI250; and Intel PVC GPU). For the details on the computational setup and for results for the lower jet multiplicities, see [8]. The evaluation on the Skylake CPU utilizes all 56 cores. We find speedups of about a factor of thirty when comparing the highest throughput on the H100 GPU, compared to the lowest throughputs on the Skylake CPU.

4 Discussion

PEPPER provides excellent unweighting efficiency via a portable implementation of the CHILI phase-space generator, and very good single-threaded event generation throughput for the heavy-hitter processes $pp \to e^+e^- + n\,\text{jets}$ and $pp \to t\bar{t} + n\,\text{jets}$ at tree level. As a portable code building on the KOKKOS portability framework it can be compiled for multi-threaded evaluation on CPU and for highly parallel evaluation using GPU acceleration. All components of the code but the eventual event output are parallelized and execute on the GPU device if available, providing speed-up factors of e.g. up to 30 when compared to multi-threaded evaluation on a CPU only. It thus eliminates the identified main bottleneck of tree-level event generation when generating particle-level SHERPA samples for the above processes, since the parton-level event generation can be completely offloaded to PEPPER via SHERPA's LHEH5 read-in capability [24]. This strategy would reduce the computational footprint by about a factor of three for these processes.

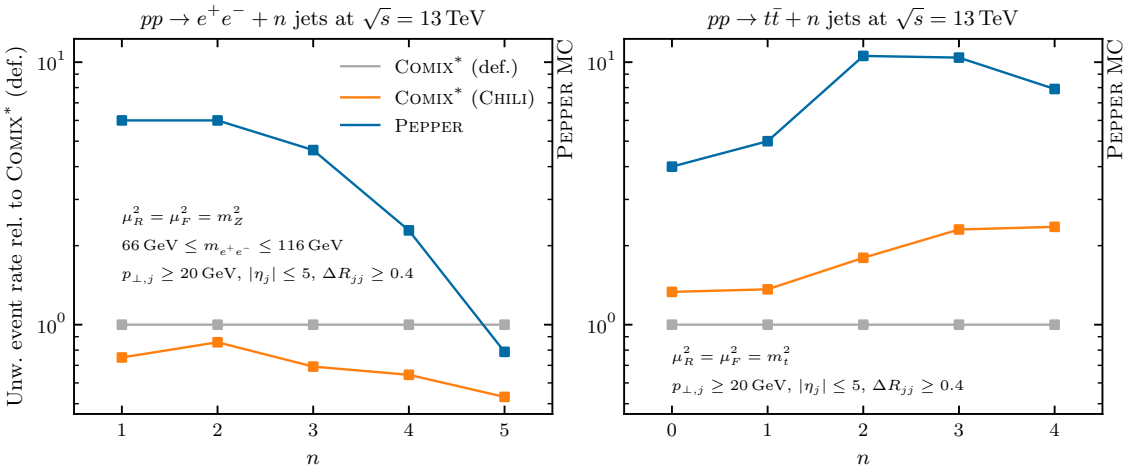

Figure 2: Single-threaded CPU unweighted event generation rates of Pepper, comparing to Sherpa using its default recursive phase-space generator ("def.") and a Sherpa-internal implementation of Chili. Figure taken from [8].

Note that [8] also studies Pepper's MPI scaling behaviour when using up to 512 A100 GPU, and presents preliminary results for using portable CPU vector instructions. In [29], a study utilizing Pepper shows that the numerical stability of the matrix-element evaluation near the infrared limit can be greatly enhanced by the novel algorithms presented therein, reaching a higher precision with double precision arithmetics compared to a naive quadruple precision implementation, at a smaller computational cost. GPU acceleration combined with this improved numerical stability opens a new avenue for efficient NNLO calculations, as the computational footprint of the expensive tree-level parts (real-real emissions) is strongly reduced by these techniques, and makes ad-hoc or expensive countermeasures for numerical instabilities like outlier rejection, artificial infrared cut-offs, or higher-precision rescue systems unnecessary.

Finally, we note that Chili and Pepper provide new opportunities for ML applications to phase-space generation [30, 31, 32, 33, 34, 35, 16] and matrix-element surrogates [36, 37]. Chili's simple structure with a very small number of channels reduces the dimensionality for learning efficient phase-space mappings, and the availability of the evaluation of both the phase-space and the matrix elements on a GPU device greatly improves the speed of training new models, compared to single-threaded evaluations on a CPU.

5 Acknowledgements

This research was supported by the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE–AC02–07CH11359. The work of M.K. and J.I. was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC-5) program, grant "NeuCol". This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility under Contract DE-AC02-06CH11357. The work of T.C. and S.H. was supported by the DOE HEP Center for Computational Excellence. E.B. and M.K. acknowledge support from BMBF (contract 05H21MGCAB). Their research is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 456104544; 510810461. E.B. acknowledges conference travel support by the Universitätsbund Göttingen e.V. This work used computing resources of the Emmy HPC system provided by The North-German Supercomputing Alliance (HLRN). M.K. wishes to thank the Fermilab Theory Division for hospitality during the final stages of this project. This research used the Fermilab Wilson Institutional Cluster and computing resources provided by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory. We are grateful to James Simone for his support.

References

[1] **HSF Physics Event Generator WG** Collaboration, S. Amoroso *et al.*, "Challenges in Monte Carlo Event Generator Software for High-Luminosity Lhc," *Comput. Softw. Big Sci.* **5** no. 1, (2021) 12, arXiv:2004.13687 [hep-ph].

[2] **HSF Physics Event Generator WG** Collaboration, E. Yazgan *et al.*, "Hl-Lhc Computing Review Stage-2, Common Software Projects: Event Generators," arXiv:2109.14938 [hep-ph].

[3] **ATLAS** Collaboration, G. Aad *et al.*, "Modelling and Computational Improvements to the Simulation of Single Vector-Boson Plus Jet Processes for the Atlas Experiment," *JHEP* **08** (2022) 089, arXiv:2112.09588 [hep-ex].

[4] **Sherpa** Collaboration, E. Bothmann *et al.*, "Event Generation with Sherpa 2.2" *SciPost Phys.* **7** no. 3, (2019) 034, arXiv:1905.09127 [hep-ph].

[5] E. Bothmann, A. Buckley, I. A. Christidi, C. Gütschow, S. Höche, M. Knobbe, T. Martin, and M. Schönherr, "Accelerating LHC event generation with simplified pilot runs and fast PDFs," *Eur. Phys. J. C* **82** no. 12, (2022) 1128, arXiv:2209.00843 [hep-ph].

[6] P. Draggiotis, R. H. P. Kleiss, and C. G. Papadopoulos, "On the Computation of Multigluon Amplitudes," *Phys. Lett. B* **439** (1998) 157–164, arXiv:hep-ph/9807207.

[7] S. Höche, S. Prestel, and H. Schulz, "Simulation of Vector Boson Plus Many Jet Final States at the High Luminosity LHC," *Phys. Rev. D* **100** no. 1, (2019) 014024, arXiv:1905.05120 [hep-ph].

[8] E. Bothmann, T. Childers, W. Giele, S. Höche, J. Isaacson, and M. Knobbe, "A Portable Parton-Level Event Generator for the High-Luminosity LHC," arXiv:2311.06198 [hep-ph].

[9] S. Hageboeck, T. Childers, W. Hopkins, O. Mattelaer, N. Nichols, S. Roiser, J. Teig, A. Valassi, C. Vuosalo, and Z. Wettersten, "Madgraph5_aMC@NLO on GPUs and vector CPUs Experience with the first alpha release," *EPJ Web Conf.* **295** (2024) 11013, arXiv:2312.02898 [physics.comp-ph].

[10] S. Carrazza, J. Cruz-Martínez, M. Rossi, and M. Zaro, "Madflow: Automating Monte Carlo Simulation on Gpu for Particle Physics Processes," *Eur. Phys. J. C* **81** no. 7, (2021) 656, arXiv:2106.10279 [physics.comp-ph].

[11] M. H. Seymour and S. Sule, "An Algorithm to Parallelise Parton Showers on a Gpu," arXiv:2403.08692 [hep-ph].

[12] J. Andersen *et al.*, "Les Houches 2023: Physics at TeV Colliders: Standard Model Working Group Report," in *Physics of the TeV Scale and Beyond the Standard Model: Intensifying the Quest for New Physics*. 6, 2024. arXiv:2406.00708 [hep-ph].

[13] H. Carter Edwards , Christian R. Trott , Daniel Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *JPDC* **74** (2014) 3202–3216.

[14] C. R. Trott, D. Lebrun-Grandié, *et al.*, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems* **33** no. 4, (2022) 805–817.

[15] N. C. . affiliates, "CUDA C++ Programming Guide," https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2024. [Online; accessed 22-July-2024].

[16] E. Bothmann, T. Childers, W. Giele, F. Herren, S. Höche, J. Isaacson, M. Knobbe, and R. Wang, "Efficient Phase-Space Generation for Hadron Collider Event Simulation," *SciPost Phys.* **15** no. 4, (2023) 169, arXiv:2302.10449 [hep-ph].

[17] G. P. Lepage, "A New Algorithm for Adaptive Multidimensional Integration," *J. Comput. Phys.* **27** (1978) 192.

[18] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr, and G. Watt, "LHAPDF6: parton density access in the LHC precision era," *Eur. Phys. J. C* **75** (2015) 132, arXiv:1412.7420 [hep-ph].

[19] F. A. Berends and W. T. Giele, "Recursive Calculations for Processes with N Gluons," *Nucl. Phys. B* **306** (1988) 759–808.

[20] E. Bothmann, W. Giele, S. Höche, J. Isaacson, and M. Knobbe, "Many-Gluon Tree Amplitudes on Modern Gpus: a Case Study for Novel Event Generators," *SciPost Phys. Codeb.* **2022** (2022) 3, arXiv:2106.06507 [hep-ph].

[21] E. Bothmann, J. Isaacson, M. Knobbe, S. Höche, and W. Giele, "QCD tree amplitudes on modern GPUs: A case study for novel event generators," *PoS* **ICHEP2022** (11, 2022) 222.

[22] A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski, and A. Verbytskyi, "The HepMC3 event record library for Monte Carlo event generators," *Comput. Phys. Commun.* **260** (2021) 107310, arXiv:1912.08005 [hep-ph].

[23] J. R. Andersen *et al.*, "Les Houches 2013: Physics at TeV Colliders: Standard Model Working Group Report," arXiv:1405.1067 [hep-ph].

[24] E. Bothmann, T. Childers, C. Gütschow, S. Höche, P. Hovland, J. Isaacson, M. Knobbe, and R. Latham, "Efficient precision simulation of processes with many-jet final states at the LHC," *Phys. Rev. D* **109** no. 1, (2024) 014013, arXiv:2309.13154 [hep-ph].

[25] The HDF Group, "Hierarchical Data Format, version 5,". https://www.hdfgroup.org/HDF5/.

[26] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, "An introduction to PYTHIA 8.2" *Comput. Phys. Commun.* **191** (2015) 159–177, arXiv:1410.3012 [hep-ph].

[27] C. Bierlich *et al.*, "A Comprehensive Guide to the Physics and Usage of Pythia 8.3" *SciPost Phys. Codeb.* **2022** (2022) 8, arXiv:2203.11601 [hep-ph].

[28] T. Gleisberg and S. Höche, "Comix, a New Matrix Element Generator," *JHEP* **12** (2008) 039, arXiv:0808.3674 [hep-ph].

[29] E. Bothmann, J. M. Campbell, S. Höche, and M. Knobbe, "Algorithms for numerically stable scattering amplitudes," arXiv:2406.07671 [hep-ph].

[30] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, "Exploring phase space with Neural Importance Sampling," *SciPost Phys.* **8** no. 4, (2020) 069, arXiv:2001.05478 [hep-ph].

[31] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, "Applying Neural Importance Sampling to gluon scattering," *PoS* **LHCP2020** (2021) 056.

[32] C. Gao, S. Höche, J. Isaacson, C. Krause, and H. Schulz, "Event Generation with Normalizing Flows," *Phys. Rev. D* **101** no. 7, (2020) 076002, arXiv:2001.10028 [hep-ph].

[33] T. Heimel, R. Winterhalder, A. Butter, J. Isaacson, C. Krause, F. Maltoni, O. Mattelaer, and T. Plehn, "Madnis - Neural Multi-Channel Importance Sampling," *SciPost Phys.* **15** no. 4, (2023) 141, arXiv:2212.06172 [hep-ph].

[34] T. Heimel, N. Huetsch, F. Maltoni, O. Mattelaer, T. Plehn, and R. Winterhalder, "The Madnis Reloaded," arXiv:2311.01548 [hep-ph].

[35] D. Yallup, T. Janßen, S. Schumann, and W. Handley, "Exploring phase space with Nested Sampling," *Eur. Phys. J. C* **82** (2022) 8, arXiv:2205.02030 [hep-ph].

[36] T. Janßen, D. Maître, S. Schumann, F. Siegert, and H. Truong, "Unweighting multijet event generation using factorisation-aware neural networks," *SciPost Phys.* **15** (2023) 107, arXiv:2301.13562 [hep-ph].

[37] D. Maître and H. Truong, "One-loop matrix element emulation with factorisation awareness," arXiv:2302.04005 [hep-ph].