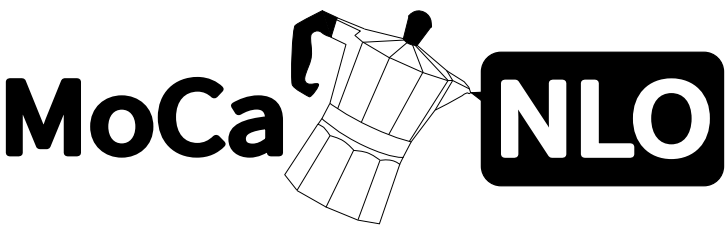
User's Guide

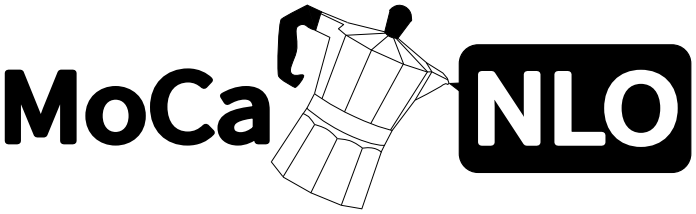
MoCaNLO

Version 1.0.0

Ansgar Denner,
Daniele Lombardi,
Santiago Lopez Portillo Chavez,
Mathieu Pellen,
Giovanni Pelliccioli

February 24, 2026





COMETA-2026-03
FR-PHENO-26-04

MoCaNLO: a Monte Carlo integrator for NLO calculations

A. Denner,[1] D. Lombardi,[2] S. Lopez Portillo Chavez,[1] M. Pellen,[3] and G. Pelliccioli[4]

[1] *Institut für Theoretische Physik und Astrophysik, Universität Würzburg, Emil-Hilb-Weg 22, 97074 Würzburg, Germany*

[2] *Dipartimento di Fisica, Università di Torino and INFN, Sezione di Torino, Via P. Giuria 1, 10125 Torino, Italy*

[3] *Physikalisches Institut, Albert-Ludwigs-Universität Freiburg, Hermann-Herder-Str. 3, 79104 Freiburg, Germany*

[4] *Dipartimento di Fisica "Giuseppe Occhialini", Università degli Studi di Milano–Bicocca and INFN, Sezione di Milano–Bicocca, Piazza della Scienza 3, 20126 Milano, Italy*

Abstract:

We present the Monte Carlo integration code MoCaNLO, which computes cross sections and distributions for processes at high-energy colliders like the LHC at leading and next-to-leading order (NLO) in the strong and electroweak couplings. It relies on the Recola package for the calculation of matrix elements and uses Catani–Seymour dipole subtraction for the treatment of infrared singularities. It has been used for several cutting-edge calculations of NLO QCD and electroweak corrections over the last years, such as NLO QCD corrections to off-shell top–antitop-quark production in association with a pair of bottom quarks and NLO electroweak corrections to vector-boson scattering processes.

February 2026

Contents

1. Introduction

The analysis of particle-physics scattering experiments relies heavily on simulations based on first-principles calculations within the framework of the Standard Model (SM) of particle physics. To this end, event generators like MadGraph5_aMC@NLO [1], Powheg-Box [2], Sherpa [3], Herwig [4], or Pythia [5] are commonly used, either as stand-alone tools or in combination, to simulate complete events from the hard scattering to hadronisation. On the other hand, for processes with many particles in the final state, these event generators are not yet able to incorporate state-of-the-art predictions as obtained in fixed-order calculations. This kind of predictions can be accomplished via dedicated Monte Carlo (MC) integration programs that often provide results in terms of weighted events. One such program is MoCaNLO, which is described in this manual.

MoCaNLO (MOnte CArlo at NLO accuracy) is a flexible multi-purpose MC integration program. It computes integrated cross sections and differential distributions for arbitrary processes at the LHC with both NLO QCD and electroweak (EW) accuracy in the SM. Thereby, full off-shell effects and spin correlations are taken into account. Besides, NLO corrections to processes at lepton–lepton and lepton–hadron colliders, as well as ultra-peripheral collisions of photons from hadrons or heavy ions are also supported.

The phase-space integration uses an elaborated multi-channel importance sampling along the lines of Refs. [6–9]. The required tree-level and one-loop matrix elements are provided by the matrix-element generator Recola [10, 11], which relies on the Collier library [12] to numerically evaluate the one-loop scalar and tensor integrals. The subtraction of the infrared (IR) divergences appearing in the real and virtual corrections has been automated based on the Catani–Seymour dipole formalism for both QCD and QED [13–18]. For the separation of photons and jets in the final state both Frixione isolation [19] and the photon-to-jet fragmentation function [20–22] are available. The IR singularities arising from the splitting of virtual photons into quark–antiquark pairs are regularised using the photon-to-jet conversion function [23]. Unstable particles are treated in the complex-mass scheme [7, 24–26]. Alternatively, the pole approximation, which is based on the pole expansion [26–28], can be used for the virtual corrections following the strategy of Ref. [29]. The corresponding non-factorisable virtual corrections are implemented according to Refs. [30–32]. For a class of polarised and unpolarised processes, the pole approximation can be employed for all contributions to NLO cross sections, while ensuring that all matrix elements are evaluated in the same Lorentz reference frame. For leading-order (LO) processes, MoCaNLO can deal with arbitrary particles in the final state. However, for NLO calculations external particles that emit real radiation, i.e. photons or gluons, have to be massless. While the light quarks are massless throughout, the bottom quark and the τ lepton can be massive, if they do not appear as radiating external particles. Masses for the electron and the muon only play a role in the regularisation of collinear singularities for initial-state

leptons or as input for the lepton structure functions, i.e. they are exclusively relevant for processes at lepton colliders.

MoCaNLO has demonstrated its ability to compute NLO corrections for high-multiplicity processes up to $2 \to 8$. Thus, it has been used to calculate NLO QCD and EW corrections to top-pair and associated top-pair production [33–41], to the associated production of single bosons [23, 42], to the production of gauge-boson pairs [43, 44], to the production of single Higgs bosons and Higgs-boson pairs via vector-boson fusion [45, 46], to triple-vector-boson production [47, 48], to vector-boson scattering (VBS) processes and their irreducible background [49–57], and to single-top processes [58]. It has also been employed to evaluate NLO QCD and EW corrections for processes involving polarised vector bosons [59–69].

This manual is structured as follows: The installation of the code is described in Sec. 2, while its basic use is illustrated in Sec. 3. The following sections discuss the input cards needed to run the program, namely the `run_card.xml` in Sec. 4, the `proc_card.xml` in Sec. 5, the `param_card.xml` in Sec. 6, the `cut_card.xml` in Sec. 7, and the `plot_card.xml` in Sec. 8. User-defined features are described in Sec. 9. After the conclusions, Sec. 10, we provide a list of validated processes in Appendix A.

2. Installation and compilation

MoCaNLO can be downloaded from the git repository

`https://mocanlo.gitlab.io`

Unpacking the tarball creates five directories. The folder `MoCaNLO` contains the actual code, while the directory `Tests` offers some test runs with corresponding sample input and output. In the folder `validated_processes`, which is referred to as `/path/to/processes` in the following, further example processes can be found. The `manual` directory contains the PDF file for the code documentation. The directory `card_generator` contains the independent package MCG, which may be used to create input cards for MoCaNLO.

The MoCaNLO package consists of the MC code itself and the additional program MCG. The MC code is independent of the program MCG, whose use is optional. For details about the usage and compilation of MCG, we refer to the file `card_generator/mcg/README.md`.

The `MoCaNLO` folder, which is referred to as `/path/to/mocanlo`, has the following subdirectory structure:

```
/path/to/mocanlo/
├── bin/
├── build/
├── include/
└── src/
```

The subdirectory `bin` contains binaries upon successful compilation and scripts for collecting results and generating plots. The `build` directory contains compiled object (`.o`) and Fortran module files (`.mod`) upon compilation. The `include` directory holds include files containing several preprocessor switches. Finally, the source code of MoCaNLO as well as some supplementary tools are placed in the subdirectory `src`.

Software requirements	
	recommended
gfortran	$\geq 8.3.0$
CMake	$\geq 3.13.4$
gnuplot	$\geq 5.4.1$

External packages	
	recommended
LHAPDF	$\geq 6.0.5$
Recola	≥ 1.5
Collier	$\geq 1.2.9$

Table 1: Software requirements and external packages needed to run MoCaNLO.

Prerequisites and external packages The software requirements for the MC code are listed in Table 1. Additionally, some external packages are needed. Each package has to be compiled separately and its installation location has to be properly added to the compilation script `compile_mocanlo` available in the `/path/to/mocanlo` folder. They are:

- **LHAPDF**: MoCaNLO is linked to LHAPDF [70] for the evaluation of parton distribution functions (PDFs). For a successful build, compile LHAPDF dynamically (`https://lhapdf.hepforge.org`) and set `LHAPDF_PATH` in the script `compile_mocanlo` to the location of the LHAPDF installation, i.e. the folder with the subdirectories `bin`, `include`, `lib` (containing `libLHAPDF.so`), and `share`. Set the LHAPATH environment variable to the path where PDF sets are installed, e.g. execute

  ```
  export LHAPATH=/path/to/lhapdf_installation/share/LHAPDF
  ```

  if these are stored in `/path/to/lhapdf_installation/share/LHAPDF`.

- **Recola/Collier**: MoCaNLO relies on Recola [10, 11] and Collier [12] for the matrix-element computation and loop-integral reduction and evaluation. The two packages have to be compiled separately from MoCaNLO. Instructions and downloads can be found at `https://collier.hepforge.org/index.html` and `https://recola.gitlab.io/index.html`. After a successful installation, the path variables `COLLIER_ROOT` and `RECOLA_ROOT` in the script `compile_mocanlo` have to be set to the location of the compiled Collier and Recola libraries, respectively.

  Recola version

  The present version of MoCaNLO does not support Recola2, which is not simply an improved version of Recola.

- **gamma-UPC** (optional): For photon–photon ultra-peripheral collisions (UPC), MoCaNLO makes use of the gamma-UPC library [71] for the photon PDF. Instructions and downloads can be found at `https://www.lpthe.jussieu.fr/~hshao/gammaupc.html`. After a successful installation, the path variable `UPCDIR` in the script `compile_mocanlo` has to be set to the location of the compiled gamma-UPC library. In addition, in the script `compile_mocanlo`, `cmake` must be executed with the option `-DUPC=ON`.

Compilation To build MoCaNLO, simply execute the script `compile_mocanlo` as:

```
./compile_mocanlo
```

The main MoCaNLO executable `mocanlo` appears in `/path/to/mocanlo/bin` upon successful compilation of the source. The script binds the different libraries to `mocanlo` using CMake. To make use of all scripts and executable (see description later), add the MoCaNLO `bin` directory to `PATH`, by executing

```
export PATH=$PATH:/path/to/mocanlo/bin
```

Tests The correct functionality of the code can be verified upon executing the test runs in the directory `Tests`. Each subdirectory *test* corresponds to a physical process and defines a test based on a single partonic subprocess. Available test folders are: `sl-vbs` ($pp \to \mu^+\nu_\mu jjjj$), `ttxz` ($pp \to e^+\nu_e\mu^-\bar{\nu}_\mu\tau^+\tau^- j_b j_b$), `tzj` ($pp \to e^+e^-\mu^+\nu_\mu j_b j$), `vbs-zz` ($pp \to e^+e^-\mu^+\mu^- jj$), and `wz-pol` ($pp \to e^+\nu_e\mu^+\mu^-$ with polarised Z and W $^+$ bosons), where j_b denotes bottom jets. The script `Tests/check_process.sh`, which must be executed inside `Tests/`, takes one of the test directories *test* as its argument

```
./check_process.sh test
```

and runs MoCaNLO using *test*`/cards` and *test*`/inputs` as input. The standard output of MoCaNLO is printed to the screen and saved in the file *test*`/output_local`, which is compared to *test*`/output`. If the standard output is as expected, the script compares all cross sections and histograms calculated locally, which are written to *test*`/test/runs/.../run_.../results`, to their counterparts found in *test*`/results`. After each pair of files is compared, the script warns if these differ and offers their location for manual comparison. The test finishes with a summary of failures. If no argument is passed to `check_process.sh`, all directories are tested.

Tests need approriate PDFs installed

Note that running the tests requires the relevant PDFs, as specified in the corresponding input cards, to be installed.

3. Getting started

3.1. Citation

If you use MoCaNLO, please cite the present manual. As mentioned above, input cards can be generated for arbitrary processes using MCG. Alternatively, example cards for a large variety of processes are provided. If you happen to use some of the example cards, please also cite the related references as indicated in the README.txt file of each process or as reported in Sec. A.

3.2. Running the example process

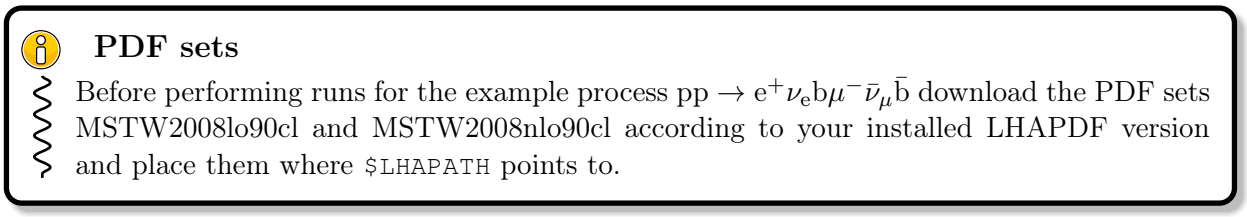
This section outlines the procedure for running MC simulations of the process $pp \to e^+\nu_e b\mu^-\bar{\nu}_\mu\bar{b}$, which is one of the examples provided within the MoCaNLO package and can be found under `/path/to/processes/ttbar/pp_tt/`. Before any MC run is submitted, the process directory `pp_epvemumvmxbbx_qcd` contains only one folder named `cards`, with four XML files that define subprocesses, standard model and MC parameters, cuts to apply, and plots to create:

```
<runs directory="ttx">
   <run id="1">
      <type>                born          </type>
      <subdirectory>        gg/born       </subdirectory>
      <subprocess           id="gg"       />
      <run_parameters       id="ttx_nlo"  />
      <model_parameters     id="ttx_nlo"  />
      <recombinations       id="ttx"      />
      <resonances           id="none"     />
      <cuts                 id="ttx"      />
      <histograms           id="default"  />
      <scales               id="ttx"      />
   </run>
   <run id="2">
      ...
</runs>
```

Listing 1: Run definition in `run_card.xml`.

`proc_card.xml`, `param_card.xml`, `cut_card.xml`, and `plot_card.xml`, explained in detail in Sec. 4.

One additional XML file, i.e. `run_card.xml`, defines MC runs for individual contributions of partonic subprocesses: In our example process one finds Born, virtual, real, and integrated-dipole contributions (see Sec. 5.1) labelled by integers.



3.3. Running a single contribution to a subprocess

The run with `id="1"`, as displayed in Listing 1, specifies the Born contribution of the gluon–gluon-induced subprocess gg → e$^+\nu_e$b$\mu^-\bar{\nu}_\mu\bar{\text{b}}$ (defined in `proc_card.xml`). This contribution constitutes the Born part of the NLO cross section.

To perform the run, execute MoCaNLO with the (relative or absolute) path to the process as first and the desired run ID as second command-line argument:

```
mocanlo /path/to/processes/ttbar/pp_tt/pp_epvemumvmxbbx_qcd 1
```

In the following, we make use of `/path/to/process = /path/to/processes/ttbar/pp_tt` to ease notations. The run prints information about the initialisation and the input parameters, and, once Recola is initialised, it outputs a list of parameters as being used for the evaluation of the matrix elements. After the initialisation phase, the MC run begins, printing intermediate result summaries once 1000 accepted events have been reached and at increasing intervals thereafter. Each time an intermediate result is printed to the terminal, this result as well as all plots defined in `plot_card.xml` are dumped to disc. Thus, the process may be killed with `Ctrl+C` without loosing the obtained results.

The run creates a new subdirectory structure in the process directory, e.g.:

```
/path/to/process/pp_epvemumvmxbbx_qcd/ttx/runs/gg/born/run_2025-06-18T15h56m45s045
```

where `ttx` is defined by `<runs directory="ttx">` in Listing 1, grouping several runs into a *runs set*. The subdirectory `gg/born/` in `runs/` is set as the run's output directory by `<subdirectory>` in the run definition. Each run with the same ID creates a subdirectory in the latter including a timestamp of the form `run_YYYY-MM-DDTHHhMMmSSsSSS`, which contains the cross sections and plots in its `result` subdirectory, as well as information about the initialisation and the run phase of the MC in its `data` subdirectory. The example run is setup in such a way that it is computed for 3 variations of the renormalisation and factorisation scale (specified in `param_card.xml` in Listing 16) at the same time by rescaling the matrix elements and the PDFs. To this end, scale factors are applied to the central scales. The first scale factor, which usually equals one, is identified with the central scale, and the corresponding result is displayed as intermediate result on the terminal. The second and third scale factors, for example, correspond to a factor of 0.5 and 2, respectively. When more than one scale factor is used, the subdirectory `result` contains folders of the type `scale_factor_`*`i`* hosting the results for each scale factor. The resulting directory structure is as follows:

```
run_2025-06-18T15h56m45s045/
├── data/
│   ├── init/
│   └── run/
└── result/
    ├── scale_factor_1/
    │   ├── histograms/
    │   │   └── data/
    │   └── cross_section.dat
    ├── scale_factor_2/
    │   ├── histograms/
    │   │   └── data/
    │   └── cross_section.dat
    └── ...
```

The intermediate result as displayed on the terminal is printed to the file `cross_section.dat` for each scale factor: At the top of this file, a selection of the most important information is also reported in one line, preceded by a pipe symbol "|", to facilitate its extraction via the `grep` command. At the bottom, a summary of occurred errors and exception is printed. The folder `histograms/data` contains data files for each plot defined in `plot_card.xml` of the form `histogram_`*`type_name`*`.dat`, e.g. one has `histogram_transverse_momentum_positron.dat` for the transverse-momentum distribution of the positron.

In addition, the subdirectory `data/init/user_input` contains a set of files that reflect the inputs to the run. The file `data/init/feynman_diagrams.tex` encodes all Feynman diagrams corresponding to the constructed integration channels in LaTeX format. With the command sequence

```
latex feynman_diagrams.tex; dvips feynman_diagrams
```

a PS file is generated that shows all these diagrams, provided that the package `axodraw` [72, 73] is installed.

The subdirectory `data/run` contains the file `feynman_diagrams_survey.tex`, which encodes again all diagrams but now in the order of importance for the integration, which is complemented by the information in the file `channel_survey.dat`. The file `integration_stability`

shows the evolution of the integration error with the number of accepted events. Finally, the file `max_weight_event.dat` stores information on the event with the largest weight and the file `typical_events.dat` information on 10 events of typical weight.

Preprocessor flags

The files `include/run_flags.h` and `include/debug_flags.h` contain preprocessor flags that allow to generate additional output for debugging. The flags `MONITOR_...` in `include/run_flags.h` provide some extra output for the generation of dipoles, the initialisation of the recombination, the calculation of the minimal centre-of-mass (CM) energy, and the interface to Recola. The flags in `include/debug_flags.h` yield extensive output for debugging. Their use requires some knowledge of the code and they should not be used in runs with many events.

3.4. Averaging of multiple runs

To speed up the integration, runs may be performed in parallel and their results, i.e. integrated cross sections and differential distributions, averaged. For these runs to be independent, different seeds for the random number generator must be used. An optional third argument to `mocanlo` allows to pass a seed to the MC, while without a third argument, a fixed standard seed is chosen. The Linux shell Bash provides the function `$RANDOM` returning a pseudo-random integer on each invocation. It may be used to provide seeds to MoCaNLO to perform independent MC runs by executing, possibly more than once

```
mocanlo /path/to/process/pp_epvemumvmxbbx_qcd 1 $RANDOM
```

which is equivalent to

```
mocanlo /path/to/process/pp_epvemumvmxbbx_qcd 1 11183
```

if `$RANDOM` returns the random number 11183.

When a seed is passed to MoCaNLO, it is appended to the name of the created run folder as `run_YYYY-MM-DDTHHhMMmSSsSSS_`*seed*, e.g. :

```
/path/to/process/pp_epvemumvmxbbx_qcd/ttx/runs/gg/born/run_2025-06-19T10h18m48s164_11183
```

with the random seed 11183.

MoCaNLO comes with a tool that averages runs, i.e. integrated cross sections and differential distributions: the script `average_runs` (to be found in the `bin` folder like all scripts described in the following) averages all runs in any subfolder of a runs set directory passed as a first argument to the binary:

```
average_runs /path/to/process/pp_epvemumvmxbbx_qcd/ttx
```

It creates the folder `average` in the directory hosting the run folders, which contains a subfolder `result` with the averaged cross section in `cross_section.dat` and the averaged histograms in the folder `histograms/data`, individually for every scale factor. When the tool encounters contributions with only one run, it still creates the folder `average`, but as a symbolic link to the single run directory. For each contribution the script performs (where needed) the average of all runs with different random seeds. If runs with equal random seeds are found, the script stops. In this case the

user is required to manually intervene to decide which folders to keep and make sure that only one run per random seed exists before launching the script again. In case a contribution was restarted using the code's features described in Sec. 6.2.1, only the last run folder that was created is considered. The latter corresponds to the folder whose name contains the additional `stepN` identifier with the highest integer `N` within the contribution's folder. Note that the averaging is computed as if the separate runs were one concatenated MC run. If one wants to only obtain an average of the cross sections (without having to wait for all histograms for all scale factors to be averaged), the `average_runs` script can also be run with a second optional argument, namely `--xs-only`, that precisely skips the histogram averaging.

3.5. Performing an NLO computation

The `run_card.xml` usually defines different runs that contribute to a full NLO calculation of the hadronic process, such as $pp \to e^+\nu_e b \mu^- \bar{\nu}_\mu \bar{b}$. An NLO calculation requires at least one run for Born, virtual corrections, real corrections, and integrated dipole contributions (Sec. 5.1).

For the considered example process, after a run of a few hours one can retrieve the NLO results. Executing

```
get_results /path/to/process/pp_epvemumvmxbbx_qcd/ttx n
```

performs the run averaging for the cross sections and prints them for the nth scale factor to the terminal. If the last argument is missing, the results for all scale factors are printed.

After averaging the runs as described in Sec. 3.4, different averaged contributions can be merged with the command

```
merge_runs -i process_path -o nlo -c "born,virt,real,idip"
```

where the argument `-i` provides the target folder where averaged results for the different contributions can be found, e.g. `process_path = /path/to/process/pp_epvemumvmxbbx_qcd/ttx`. The argument `-o` specifies the name of the output folder in the `results` directory, which in the example above is located in `process_path/results/nlo`. Finally, `-c` contains the list of names of contributions to be merged, enclosed in double quotes and either space or comma separated. Note that the name of contributions in the list must match the names of the folders where the results for the different contributions for each partonic channel are written by the MoCaNLO code, i.e. as specified in the `run_card.xml`. A summary of how to execute the script can be found by typing

```
merge_runs -h
```

The script sums up all indicated contributions to cross sections and histograms into the subdirectory `nlo` of `results`. The structure of the output folder is as follows:

```
ttx/
└── results/
    └── nlo/
        ├── hadronic_cross_section.dat
        └── histograms/
            └── data/
                ├── scale_factor_1/
                ├── scale_factor_2/
                └── .../
```

The merged cross sections are listed for all scale factors in the file `hadronic_cross_section.dat` in the directory `nlo`. The merged histogram files are written to the directories `scale_factor_`*i*.

3.6. Plotting results

MoCaNLO comes with some scripts that allow to plot results using `gnuplot` based on the histograms obtained with `merge_runs`. The script `create_k-factor_plots` produces absolute distributions together with ratios of contributions, `create_correction_plots` absolute distributions along with relative contributions, and `create_scale_envelope_plots` absolute distributions and relative contributions with scale envelopes based on the results in the different `scale_factor_`*i* directories. The syntax of these commands reads:

```
create_k-factor_plots -i process_path -o k-factor_plots -c "lo,nlo"
```

The argument `-i` provides the runs set folder where merged results for the different contributions are located in the subdirectory `results`. The argument `-o` specifies the name of the output folder in the `plots` directory, which in the example above is located in `process_path/plots/k-factor_plots`. This is also the default, in case the argument `-o` is omitted. Finally, `-c` contains the list of names of contributions to be plotted, enclosed in double quotes and either space or comma separated. These contributions have to be created via `merge_runs` with the corresponding names. Thus, in the example above the subfolders `results/lo` and `results/nlo` must exist and contain the relevant information. The first contribution is used as reference for the relative plots.

A folder `plots` is created in the runs set directory, which contains the subdirectory specified in the plot command. The resulting directory structure is as follows

```
ttx/
├── results/
│   ├── lo/
│   │   └── histograms/
│   │       └── data/
│   │           ├── scale_factor_1/
│   │           ├── scale_factor_2/
│   │           └── .../
│   └── nlo/
│       └── histograms/
│           └── data/
│               ├── scale_factor_1/
│               ├── scale_factor_2/
│               └── .../
└── plots/
    └── k-factor_plots/
```

Single EPS and PDF files for the individual histograms are placed in `plots/k-factor_plots` together with a PDF file named `k-factor_plots.pdf` that collects all histograms. Each plot shows the different contributions, i.e. `lo` and `nlo` in this example, in an upper panel and the contributions divided by the first one in the lower panel, i.e. `lo/lo` and `nlo/lo` in the example.

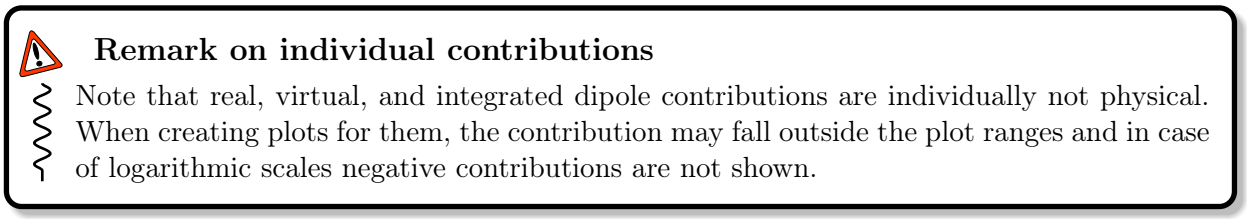
The script `create_correction_plots` works in the same way, but is meant for displaying relative corrections. If for instance the virtual, real, and integrated dipole corrections have been merged

with appropriate executions of the `merge_runs` script and stored in the folders `results/virt`, `results/real`, and `results/idip`, these could be plotted via

```
create_correction_plots -i process_path -o correction_plots \
                        -c "lo,virt,real,idip"
```

The resulting plots show the absolute contributions `lo`, `virt`, `real`, and `idip` in the upper panel and the ratios of `virt/lo`, `real/lo`, and `idip/lo` in the lower panel in per cent. The default output folder, in case the argument `-o` is omitted, is `process_path/plots/correction_plots`.

The script `create_scale_envelope_plots` works as the script `create_k-factor_plots`, but displays in addition bands for the scale variation obtained from the maxima and minima in the different `process_path/results/.../histograms/data/scale_factor_`*i* directories. It creates in `.../histograms/data` a further subdirectory called `scale_envelope` containing the data for the scale-envelope plots.



Remark on individual contributions

Note that real, virtual, and integrated dipole contributions are individually not physical. When creating plots for them, the contribution may fall outside the plot ranges and in case of logarithmic scales negative contributions are not shown.

4. Setting up a process in the `run_card.xml`

A run for a given process is fully specified in MoCaNLO by five input cards: `run_card.xml`, `proc_card.xml`, `param_card.xml`, `cut_card.xml`, and `plot_card.xml` (see Table 2). The code expects them to be located in a folder named `cards` inside the specific process directory:

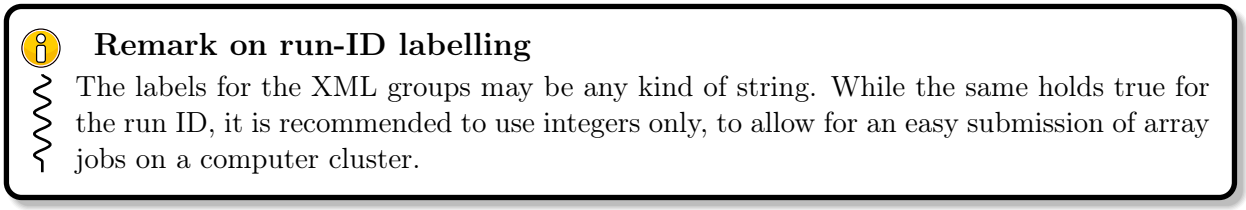
```
pp_epvemumvmxbbx_qcd/
└── cards/
    ├── run_card.xml
    ├── proc_card.xml
    ├── param_card.xml
    ├── cut_card.xml
    └── plot_card.xml
```

The `run_card.xml` plays the role of the top-level card. In it, all contributions (to be computed as individual runs), which together represent an independent and self-consistent calculation, are grouped by runs sets, as illustrated in Listing 1. In this example, the XML group `<runs directory="ttx">` defines a runs set `ttx`, which means that the output of all runs in this group will appear in subdirectories of the folder `ttx` in the process directory.

Within a runs set, the most elemental part of a calculation is represented by a run, which is fully specified by the tags contained in the XML group `<run id="N">`, with `N` an integer number. In our example, `<run id="1">` defines the process with ID 1.

Card	XML section(s)		Description	
run_card.xml	<run>	-	Definition of MC runs using links to other cards	
proc_card.xml	<subprocess>	✓	List of partonic subprocesses and their coupling order	Sec. 5.1
	<resonances>		Specification of resonances restricting the generated phase-space mappings and matrix-element contributions	Sec. 5.2
	<on_shell_projection>		Details of the pole approximation and its on-shell projection	Sec. 5.2
param_card.xml	<model_parameters>	✓	Parameters of the particle model such as masses and widths	Sec. 6.1
	<run_parameters>	✓	Various MC-run parameters	Sec. 6.2
	<weight_opts>		Tuning of the multi-channel weight optimisation	Sec. 6.3
cut_card.xml	<recombinations>		Definitions of recombinations for the jet algorithm	Sec. 7.1
	<cuts>		Phase-space cuts	Sec. 7.3
	<scales>		Definition of the renormalisation/ factorisation scale	Sec. 7.4
plot_card.xml	<histograms>		Histograms	Sec. 8

Table 2: List of cards for the process definition, together with the different XML sections they contain. The symbol ✓marks the sections that must be linked to a process definition in the run_card.xml, while all remaining ones are optional. The rightmost column contains the section of this manual that describes each section.



The tags that define a run can be divided in three different classes:

- **basic tags**: This set comprises two tags that should always be specified when defining a process. The tag <type> is used to define the type of contribution to be computed and can take one of the following values: born, real, virt, idip, lpid, vnfc, idpi, and idpk. The output and results of the run are stored in the folder's runs set in the path given by the tag <subdirectory>.
- **linking tags**: These tags are used to link XML sections that are defined in the other four cards. Each of these tags has a one-to-one correspondence with these sections, whose complete list is reported in Table 2. For instance, the tag <cuts id="ttx"> in Listing 1 refers to the homonymous XML group in cut_card.xml, which hosts a specific set of cuts. These tags are particularly useful, since they offer the flexibility to use different setups (e.g. parameters, cuts,

set of histograms, ...) for different IDs, and simplify switching back and forth between setups. Note that the linking tags need to be specified for every process: the ones that MoCaNLO requires for a consistent run are marked with the symbol ✓in Table 2. If those are not present, the code will stop with an appropriate message.

- **flag tags**: This set includes only the optional tag `<osp_type>`, which controls some features of the pole approximation as described in Sec. 5.2.

Setting default linking tags in the `run_card.xml`

All linking tags in the run card apart from `<subprocess>` can also be defined outside the definition of any `<run id="N">` group. These definitions serve as a default for all following run definitions. The defaults can be overwritten within a run definition. They can also be redefined outside a run definition, and then fix a new default for the following run definitions.

5. Definition of the process with the `proc_card.xml`

The process card (`proc_card.xml`) contains the list of all partonic processes that can be linked in the `run_card.xml` to define a process with a given run ID (see Listing 1). This list is enclosed in the XML section `<subprocesses>`. In there, each subprocess contains the information that uniquely defines the `<run_type>`, i.e. the specific LO or NLO contribution to be computed, as described in Sec. 5.1.

In the process card, intermediate resonances can be specified in the section `<resonances>` to enable a dedicated phase-space integration and/or the pole-approximation algorithm, which serves as a basis for the definition of polarised cross sections (see Sec. 5.2). The section `<on_shell_projection>` allows to control some features of the on-shell projection used to compute it.

5.1. Partonic process specification

A subprocess definition is enclosed in an XML `<subprocess>` block identified by a unique subprocess ID (see Listing 2), which is used by the `run_card.xml` to refer to it. Its definition comprises two parts: the partonic process (`<partonic_process>`), which describes the lowest-order contribution to the considered channel, and the real process (`<real_process>`), which specifies the contribution with one additional final-state particle. The same subprocess ID can be referred to in the `run_card.xml` by multiple run IDs with different `<run_type>` values. This allows to compute the various contributions to the NLO cross section of a specific channel.

The parameters of both the `<partonic_process>` and the `<real_process>` are summarised in Table 3 For both the particle content of the initial and final states is defined in the tags `<incoming>` and `<outgoing>`, respectively, using the name convention for the particles reported in the second column of Table 4. The tags `<pdf1_codes>` and `<pdf2_codes>` indicate the PDF codes (using the numbering convention in the fourth column of Table 4) to be used for the first and second incoming beam, respectively.

```
<subprocesses>
    <subprocess id="uxu">
        <partonic_process>
            <incoming>      u~ u             </incoming>
            <outgoing> e+ ve b mu- vm~ b~    </outgoing>
            <pdf1_codes>      -2             </pdf1_codes>
            <pdf2_codes>       2             </pdf2_codes>
            <tree_qcd_order>   2             </tree_qcd_order>
            <loop_qcd_order>   4             </loop_qcd_order>
        </partonic_process>
        <real_process>
            <incoming>      u~ u             </incoming>
            <outgoing> e+ ve b mu- vm~ b~ g  </outgoing>
            <pdf1_codes>      -2             </pdf1_codes>
            <pdf2_codes>       2             </pdf2_codes>
            <tree_qcd_order>   3             </tree_qcd_order>
        </real_process>
    </subprocess>
    ...
</subprocesses>
```

Listing 2: Subprocess definition for the $\mathcal{O}(\alpha_s^2\alpha^4)$ and its NLO QCD corrections for top-pair production.

parameter	possible values	default value
`<incoming>`	pair of particle names	none
`<outgoing>`	list of particle names	none
`<pdf1_codes>`	list of integers	none
`<pdf2_codes>`	list of integers	none
`<tree_qcd_order>`	integer	0
`<tree_qcd_order_2>`	integer	0
`<loop_qcd_order>`	integer	0
`<interference>`	integer	0
`<tree_e0_order>`	integer	guessed
`<loop_e0_order>`	integer	guessed

Table 3: Parameters of the `<partonic_process>` or `<real_process>` subsections of `proc_card.xml` and their default values.

The orders of the amplitude in the QCD coupling g_s are fixed in MoCaNLO with the help of additional tags. The orders in the EW coupling e follow for a given amplitude from the number of external particles. We use the usual quantities $\alpha_s = g_s^2/(4\pi)$ and $\alpha = e^2/(4\pi)$.

The value of the tag `<tree_qcd_order>` defines the power of the strong coupling g_s at the amplitude level. For the virtual contribution, which originates from the interference of tree-level and one-loop amplitudes, the QCD order is fixed by specifying separately the powers of g_s entering the tree-level and one-loop amplitudes using the `<tree_qcd_order>` and `<loop_qcd_order>` tags, respectively. An example of the usage of these tags is given in Listing 2. There, the $\mathcal{O}(\alpha_s^2\alpha^4)$ and its NLO QCD corrections to the cross section are specified for a partonic channel contributing to the example process $pp \to e^+\nu_e b \mu^- \bar{\nu}_\mu \bar{b}$.

Name	MoCaNLO	Recola	PDG code	Jet type	PDF code
d	d	d	1	1	1
u	u	u	2	1	2
s	s	s	3	1	3
c	c	c	4	1	4
b	b	b	5	2	5
t	t	t	6	7	6
e^-	e-	e-	11	5	8
e^+	e+	e+	−11	4	−8
ν_e	ve	nu_e	12	6 (21)	–
$\bar{\nu}_e$	ve~	nu_e~	−12	6 (22)	–
μ^-	mu-	mu-	13	17	9
μ^+	mu+	mu+	−13	16	−9
ν_μ	vm	nu_mu	14	6 (23)	–
$\bar{\nu}_\mu$	vm~	nu_mu~	−14	6 (24)	–
τ^-	ta-	tau-	15	19	10
τ^+	ta+	tau+	−15	18	−10
ν_τ	vt	nu_tau	16	6 (25)	–
$\bar{\nu}_\tau$	vt~	nu_tau~	−16	6 (26)	–
γ	a	A	22	3	7
g	g	g	21	1	0
Z_0	z	Z0	23	8	–
W^+	w+	W+	24	9	–
W^-	w-	W-	−24	10	–
H	h	H	25	11	–

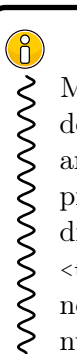
Table 4: The different columns in the table report: particle names, identifiers in MoCaNLO and Recola, the particle codes according to the PDG MC particle numbering scheme, the jet types for the definition of recombinations and cuts, and the PDF codes according to LHAPDF. Antiparticles of non-self-conjugated particles have the negative PDG and PDF code and a "~" added to the MoCaNLO and Recola identifier (except for particles with explicit charge signs in the identifier, which change for the antiparticle). Jet types are usually the same for the antiparticle, except for the charged leptons (4, 16, 18) and the W^- boson (10). The different jet type for neutrinos in parenthesis can be switched on with the input option <neutrino_species_tagging>.

For processes that receive contributions of different orders in the QCD coupling at LO, the additional tags <interference> and <tree_qcd_order_2> are needed. Contributions of interferences between LO matrix elements with different coupling orders can be obtained by setting <interference> = 1 together with <tree_qcd_order> and <tree_qcd_order_2> to the power of the strong coupling of the two interfering matrix elements. Interferences between one-loop and tree-level matrix elements

```
<subprocess id="uxu_int">
    <partonic_process>
        <incoming>      b~ b              </incoming>
        <outgoing> e+ ve b mu- vm~ b~     </outgoing>
        <pdf1_codes>      -2              </pdf1_codes>
        <pdf2_codes>       2              </pdf2_codes>
        <tree_qcd_order>   2              </tree_qcd_order>
        <tree_qcd_order_2> 0              </tree_qcd_order_2>
        <interference>     1              </interference>
        <loop_qcd_order>   2              </loop_qcd_order>
    </partonic_process>
    <real_process>
        <incoming>      b~ b              </incoming>
        <outgoing> e+ ve b mu- vm~ b~ g   </outgoing>
        <pdf1_codes>      -2              </pdf1_codes>
        <pdf2_codes>       2              </pdf2_codes>
        <tree_qcd_order>   2              </tree_qcd_order>
        <tree_qcd_order_2> 0              </tree_qcd_order_2>
        <interference>     1              </interference>
    </real_process>
</subprocess>
```

Listing 3: Subprocess definition for the $\mathcal{O}(\alpha_s\alpha^5)$ and its NLO EW corrections to top-pair production.

are obtained in an analogous way by specifying `<loop_qcd_order>` and `<tree_qcd_order_2>`, and interferences of loop-induced amplitudes by `<tree_qcd_order>` and `<tree_qcd_order_2>`. Note that when `<interference>` is not specified or set to `0`, the tag `<tree_qcd_order_2>` is ignored. This feature is needed for our example process $pp \to e^+\nu_e b\mu^-\bar{\nu}_\mu\bar{b}$, which receives at LO three different contributions from the $\mathcal{O}(\alpha_s^2\alpha^4)$, $\mathcal{O}(\alpha_s\alpha^5)$, and $\mathcal{O}(\alpha^6)$. To evaluate the $\mathcal{O}(\alpha_s\alpha^5)$, for instance, one needs to interfere amplitudes with different powers of g_s, as exemplified in Listing 3.



Using `tree_qcd_order` to tune the integration-channel generation

MoCaNLO generates integration channels on the basis of Feynman diagrams. Independently of the values of run type and `<interference>`, tree diagrams contributing to the amplitude of order `<tree_qcd_order>` are always used for the channel generation. To improve the integration of contributions originating from the interference of amplitudes of different orders, the largest possible set of integration channels can be chosen by setting `<tree_qcd_order>` to the lower of the two interfering orders. This allows to generate channels according to amplitudes with the largest power of e, which typically comprise a larger number of Feynman diagrams.

For processes with external on-shell photons, the corresponding couplings should always be renormalised at zero momentum transfer in order to avoid a dependence on the light quark masses [26]. This can be ensured by specifying the number of electromagnetic couplings $\alpha(0)$ at tree and one-loop level using the tags `<tree_e0_order>` and `<loop_e0_order>`, respectively, while the remaining electromagnetic couplings are renormalised using the scheme chosen via `<scheme_alpha>` in the `param_card.xml`. This is illustrated in Listing 4 for the partonic process $\bar{u}u \to e^+e^-\gamma$ and its EW correction. Note that in this example only the coupling of one photon is fixed to $\sqrt{\alpha(0)}$. Alternatively, also the coupling of the real-emission photon could be set to $\sqrt{\alpha(0)}$ by setting `<tree_e0_order>` $= 2$ in `<real_process>`. However, in this case one has to additionally include `<loop_e0_order>` $= 2$ in `<partonic_process>` in order not to spoil the cancellation of IR singularities.

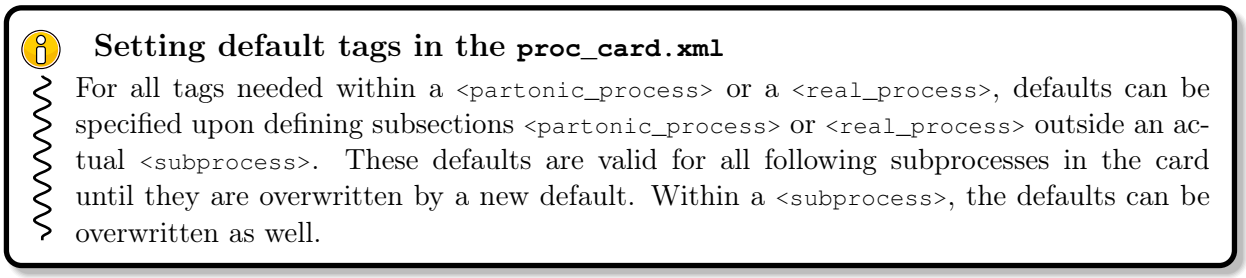
```
<subprocess id="uxu">
    <partonic_process>
        <incoming>          u~ u       </incoming>
        <outgoing>          e+ e- a    </outgoing>
        <pdf1_codes>       -2 -4       </pdf1_codes>
        <pdf2_codes>        2  4       </pdf2_codes>
        <tree_qcd_order>    0          </tree_qcd_order>
        <tree_qcd_order_2>  0          </tree_qcd_order_2>
        <loop_qcd_order>    0          </loop_qcd_order>
        <tree_e0_order>     1          </tree_e0_order>
        <loop_e0_order>     1          </loop_e0_order>
    </partonic_process>
    <real_process>
        <incoming>          u~ u       </incoming>
        <outgoing>          e+ e- a a  </outgoing>
        <pdf1_codes>       -2 -4       </pdf1_codes>
        <pdf2_codes>        2  4       </pdf2_codes>
        <tree_qcd_order>    0          </tree_qcd_order>
        <tree_qcd_order_2>  0          </tree_qcd_order_2>
        <loop_qcd_order>    0          </loop_qcd_order>
        <tree_e0_order>     1          </tree_e0_order>
    </real_process>
</subprocess>
```

Listing 4: Subprocess definition for the $\mathcal{O}(\alpha^2\alpha(0))$ and its NLO EW corrections of order $\mathcal{O}(\alpha^2\alpha(0))$ for e^+e^--pair production in association with a photon.

If `<tree_e0_order>` and/or `<loop_e0_order>` are not specified in `proc_card.xml`, MoCaNLO uses the following defaults. For processes without external photons, both are set to zero. For processes with external photons, both are set to the number of photons in the final state of `<partonic_process>`. Since the treatment of a possible bremsstrahlung photon is not unique for a real process, MoCaNLO stops and offers a guess. If this guess corresponds to the desired choice, it can be accepted by adding the tag

```
<ignore_e0_check>  true </ignore_e0_check>
```

in the `<run_parameters>` section of `param_card.xml`, which prevents the code from stopping. Note that a proper subtraction of infrared singularities requires the same value for `<tree_e0_order>` for the `<real>` and `<idip>` runs.



A standard computation at NLO accuracy (in either the α_s or the α coupling) requires to run contributions of Born (B), virtual (V), integrated-counterterm (I), and subtracted-real ($R - D$) type. More in detail, the differential cross section in a generic observable $\mathcal{O}$ at NLO reads,

$$\frac{\mathrm{d}\sigma^{\mathrm{NLO}}}{\mathrm{d}\mathcal{O}} = \int_{\mathrm{cuts}} \mathrm{d}\Phi_n \, \delta_n(\mathcal{O}) \, (B + V + I) + \int_{\mathrm{cuts}} \mathrm{d}\Phi_{n+1} \big[\delta_{n+1}(\mathcal{O}) \, R - \delta_n(\mathcal{O}) \, D\big], \tag{1}$$

where the labels n and $n+1$ indicate that cuts and the evaluation of the observable should be applied (after possible recombination and isolation) to Born-like and real-correction-like phase-space

kinematics, respectively. These four contributions are identified in MoCaNLO by the `born`, `virt`, `idip`, and `real` run types. The local subtraction counterterms (D) and the corresponding integrated counterparts (I) enable the subtraction of IR singularities of QED and QCD type. In the term dubbed I, all integrated counterparts of the local counterterms D are collected, including also initial-state collinear and PDF-renormalisation terms. In MoCaNLO the counterterms are implemented in the Catani–Seymour dipole formalism [13–18] and limited to IR-singular configurations caused by massless external particles (with the exception of NLO EW corrections to processes with intermediate W bosons in the pole approximation, as detailed in Sec. 5.2.2).

Infrared-finite, loop-induced contributions must be handled with the `lpid` run type. Other run types can be used for the calculation of contributions in the pole approximation as described in Sec. 5.2.3, specifically for the virtual non-factorisable corrections (`vnfc`), for the I-operator part of integrated counterterms (`idpi`), and for the P- and K-operator parts of the integrated dipole contributions (`idpk`).

It is important to note that, depending on the `run_type` of a given run, only some information of the `<subprocess>` block is used.

- **`born`**: this type is intended for processes that receive tree-level contributions at LO. Only the `<partonic_process>` part of the subprocess definition is used, with the order of the amplitudes computed using `tree_qcd_order` and if relevant `tree_e0_order`; when `interference` is set to `1`, `tree_qcd_order_2` is also used.

- **`virt`**: this is intended for one-loop virtual corrections to processes that receive contributions at tree level. Only the `<partonic_process>` part of the subprocess definition is used, with the order of the amplitudes computed using `loop_qcd_order` and `tree_qcd_order`, or, when `interference` is set to `1`, `tree_qcd_order_2`; if relevant also `tree_e0_order` and `loop_e0_order` enter.

- **`vnfc`**: only the `<partonic_process>` part of the subprocess definition is used, with the order of the amplitude computed as for the `born run_type`. This type is dedicated to the computation of virtual non-factorisable contributions in the pole approximation (see Sec. 5.2.3). Its usage is only allowed if the section `<resonances id="XX">` is specified in the `run_card.xml` (see Sec. 5.2). The type of non-factorisable corrections to be computed is steered by `tree_qcd_order` and `loop_qcd_order`: if `loop_qcd_order` > `tree_qcd_order` non-factorisable QCD corrections are evaluated, and if `loop_qcd_order` = `tree_qcd_order` the EW ones are selected. If `interference` = `1`, `tree_qcd_order_2`, and `loop_qcd_order` are instead compared to choose between the two types of corrections.

- **`lpid`**: this run type is intended for loop-induced processes. It works as a `born` one, where the orders of the loop diagrams are specified with `<tree_qcd_order>` (and not by `loop_qcd_order`), and, if `interference` is set to `1`, `<tree_qcd_order_2>`, thus allowing to calculate loop-induced interference contributions. Analogously, the orders of $\sqrt{\alpha(0)}$ are specified via `<tree_e0_order>`.

- **`real`**: a real contribution comprises the real squared amplitude and all Catani–Seymour subtraction counterterms needed to have a finite result. The information to define the real squared amplitude is taken from the `<real_process>` block. The same applies for the construction of its subtraction counterterms. Their complete list is obtained considering all underlying Born processes that can be reached from the real process by discarding a candidate emissus particle and replacing the candidate emitter particle by the splitting one. The underlying Born pro-

cesses whose particle content is not compatible with the single-particle-cut requirements (see Sec. 7.3) specified in the `cut_card.xml` are discarded already at this level, to prevent the computation of contributions that would eventually be cut away. Note that the PDF codes in the `<real_process>` part are also the relevant ones for the computation of the subtraction terms. Note that, when the `real` is treated in the pole approximation (see Sec. 5.2.2), the code also uses the information of the `<incoming>` and `<outgoing>` from the `<partonic_process>` block to identify the radiated particle.

- **`idip`**: for this run type both the `<partonic_process>` and the `<real_process>` parts are relevant. This contribution is computed using the PDF codes in the `<real_process>` part. It results from the sum of dipoles constructed as for the `real` case that also match the particle content (up to ordering of the final state) and the amplitude order of the underlying Born process specified in the `<partonic_process>` block. Only these dipoles are taken into account in the result. This means that the complete set of integrated counterterms is obtained by running multiple `idip` runs whose `<subprocess>` definitions have identical `<real_process>` parts, but different `<partonic_process>` as underlying Born processes. Also note that, since integrated dipoles are defined on a Born-like kinematics, the integration channels are constructed from the `<partonic_process>` amplitude.

- **`idpi`**/**`idpk`**: these two additional types can be run exactly as the idip `run_type` described above. Even if they can be used both with and without the specification to the section `<resonances id="XX">` in the `run_card.xml` (see Sec. 5.2), their purpose is to guarantee the IR safety of the full result when a pole approximation is applied only to the virtual contributions (as described in Sec. 5.2.3). Note that the contribution `idpk` cannot be calculated in the pole approximation: If it is linked to a section `<resonances id="XX">` of the `proc_card.xml` with `pole_approximation=true`, the code stops.

In Listing 5 the usage of the run types `real` and `idip` is further illustrated. The example shows some subprocesses needed to compute QCD corrections of $\mathcal{O}(\alpha_s\alpha^6)$ to the pure EW LO contribution to WZ VBS. All three subprocesses with IDs `real_b1`, `real_b2`, and `real_b3` define the same real squared amplitude, but differ in their `<partonic_process>` block. The real contribution to the channel uu $\to e^+\nu_e\mu^+\mu^-$udg is obtained by running only one of the three subprocesses with run type `real`. To properly account for all integrated dipoles, instead all three subprocesses must be run separately with run type `idip`. Then, the subprocess with `id="real_b1"` computes integrated QCD dipoles with underlying Born of order `<tree_qcd_order>` $= 0$, while the subprocesses with `id="real_b2"` and `id="real_b3"` compute QED dipoles with two different underlying Born processes (resulting from a photon emitted from the incoming quark and entering the hard process) of order `<tree_qcd_order>` $= 1$.

5.1.1. Merging of partonic channels

Having separate runs for different partonic channels and NLO contributions allows MoCaNLO to tune the integration channels and improve the integration efficiency. This is crucial for processes having a non-trivial resonance structure. Nonetheless, for high-multiplicity final states, the number of contributions to be computed as separate runs grows very fast and can potentially become a bottleneck. To ameliorate this problem, MoCaNLO supports two ways of computing different partonic channels together, which may be used simultaneously.

```
<subprocess id="real_b1">
   <partonic_process>
      <incoming> u  u    </incoming>
      <outgoing> e+ ve  mu+ mu- u  d    </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 0 </tree_qcd_order>
      <loop_qcd_order> 2 </loop_qcd_order>
   </partonic_process>
   <real_process>
      <incoming> u  u    </incoming>
      <outgoing> e+ ve  mu+ mu- u  d  g </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 1 </tree_qcd_order>
   </real_process>
</subprocess>
<subprocess id="real_b2">
   <partonic_process>
      <incoming> a  u    </incoming>
      <outgoing> e+ ve  mu+ mu- d  g    </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 1 </tree_qcd_order>
      <loop_qcd_order> 0 </loop_qcd_order>
   </partonic_process>
   <real_process>
      <incoming> u  u    </incoming>
      <outgoing> e+ ve  mu+ mu- u  d  g </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 1 </tree_qcd_order>
   </real_process>
</subprocess>
<subprocess id="real_b3">
   <partonic_process>
      <incoming> u  a    </incoming>
      <outgoing> e+ ve  mu+ mu- d  g    </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 1 </tree_qcd_order>
      <loop_qcd_order> 0 </loop_qcd_order>
   </partonic_process>
   <real_process>
      <incoming> u  u    </incoming>
      <outgoing> e+ ve  mu+ mu- u  d  g </outgoing>
      <pdf1_codes>     2 </pdf1_codes>
      <pdf2_codes>     2 </pdf2_codes>
      <tree_qcd_order> 1 </tree_qcd_order>
   </real_process>
</subprocess>
```

Listing 5: Example of three subprocess definitions in the `proc_card.xml` needed to compute different integrated-dipole contributions to the same real channel.

PDF merging The tags `<pdf1_codes>` and `<pdf2_codes>`, which define the PDF codes for the first and second beam, can be initialised to strings of integers, like $\{i_1, \ldots, i_n\}$ and $\{j_1, \ldots, j_m\}$, respectively. Their length should be the same for both tags (i.e. $n = m$), since values occupying the same positions in the string enter the calculation in pairs, namely only the combinations $\{(i_1, j_1), \ldots, (i_n, j_n)\}$ are considered and no cross terms. The resulting cross section will be the sum of as many contributions as the length of the strings. Each contribution is obtained by weighting the same squared amplitude with different PDF factors. The underlying assumption is that ampli-

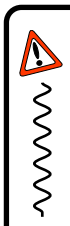
```
<subprocess id="uux_fs_merged">
    <partonic_process>
        <incoming>          u u~          </incoming>
        <outgoing> e+ ve mu+ mu- u~ d     </outgoing>
        <outgoing> e+ ve mu+ mu- c~ s     </outgoing>
        <pdf1_codes>        2  4          </pdf1_codes>
        <pdf2_codes>       -2 -4          </pdf2_codes>
        <tree_qcd_order>    0             </tree_qcd_order>
        <tree_qcd_order_2>  0             </tree_qcd_order_2>
        <interference>      1             </interference>
    </partonic_process>
    <real_process>
        <incoming>          u u~          </incoming>
        <outgoing> e+ ve mu+ mu- u~ d a   </outgoing>
        <outgoing> e+ ve mu+ mu- c~ s a   </outgoing>
        <pdf1_codes>        2  4          </pdf1_codes>
        <pdf2_codes>       -2 -4          </pdf2_codes>
        <tree_qcd_order>    0             </tree_qcd_order>
        <tree_qcd_order_2>  0             </tree_qcd_order_2>
        <interference>      1             </interference>
    </real_process>
</subprocess>
```

Listing 6: Example of channel combination using PDF and final-state merging.

tudes for the considered process are identical when exchanging the flavours $i_1 \leftrightarrow i_k$ and $j_1 \leftrightarrow j_k$ $\forall k \in \{2, \ldots, n\}$ both in the initial and in the final states.

The lines containing `<pdf1_codes>` and `<pdf2_codes>` in Listing 6 illustrate the usage of PDF merging for partonic processes contributing to W^+Z VBS. In the example the partonic channels $u\bar{u} \to e^+\nu_e\mu^+\mu^-\bar{u}d(\gamma)$ and $c\bar{c} \to e^+\nu_e\mu^+\mu^-\bar{c}s(\gamma)$ are computed in one run. This combination is allowed, since the two processes only differ by PDF factors, but have identical matrix elements if the quarks are massless and the CKM matrix is set to the identity matrix.

Final-state merging If the definition of a `<partonic_process>`/`<real_process>` includes multiple `<outgoing>` tags, partonic channels sharing the same initial state and only differing in their final states are merged into one MoCaNLO run. When the process is initialised, the code generates all integration channels for the different processes and eliminates identical ones. For the `born` and `virtual` types, the list of processes in the `<partonic_process>` block is combined, while for the `real` type only the list in `<real_process>` is relevant. For the `idip` type, both `<partonic_process>` and `<real_process>` are important. For each real process at a given position in the list of `<outgoing>` tags of `<real_process>`, the code computes only the dipoles which match the underlying Born at the same position in the `<outgoing>` tags of `<partonic_process>`. Both when running the `real` and the `idip`, identical dipoles and dipoles only differing in some charge/colour factors are computed once and properly accounted for in the combined result.



Warning

The final-state merging is limited to channels that differ in the flavour of final-state fermions. No merging is supported if a fermion is replaced by a gluon or a photon. Further, it is implicitly assumed that recombination and cuts act on all merged final states in exactly the same way. The reason is that recombination rules and cuts are only evaluated

```
<resonances id="ttbar">
   <resonance>
      <particle>       t      </particle>
      <external_legs>  3 4 5  </external_legs>
    </resonance>
    <resonance>
      <particle>       t~     </particle>
      <external_legs>  6 7 8  </external_legs>
   </resonance>
</resonances>
```

Listing 7: Example of a minimal specification of resonances corresponding to the partonic process defined in Listing 2 or 3.

for the first of the merged processes. Thus, normally quarks should not be merged with charged leptons or neutrinos, since typically cuts act differently on these final-state objects.

Merging final states efficiently

MoCaNLO generates integration channels on the basis of Feynman diagrams for all merged partonic processes and eliminates duplicate ones. In order to reduce the number of channels, the final states of the different partonic processes should be ordered in such a way that as many diagrams as possible become identical. This is typically achieved by putting corresponding particles in the same position in different final states.

In Listing 6 the syntax for integrating together the channels $u\bar{u} \to e^+\nu_e\mu^+\mu^-\bar{u}d(\gamma)$ and $u\bar{u} \to e^+\nu_e\mu^+\mu^-\bar{c}s(\gamma)$ is explicitly shown. Since in this example the final-state merging is used in combination with the PDF merging, the channels $c\bar{c} \to e^+\nu_e\mu^+\mu^-\bar{c}s(\gamma)$ and $c\bar{c} \to e^+\nu_e\mu^+\mu^-\bar{u}d(\gamma)$ are also implicitly accounted for.

5.2. Treatment of resonances

Many applications focus on contributions involving a specific set of intermediate resonances for a given process. The user can require a desired set of resonances by linking in the `run_card.xml` a section `<resonances id="XX">`, where `XX` is the ID that denotes a particular resonance specification. This optional section must be provided in the `proc_card.xml` as shown in Listing 7.

In each resonances section, multiple resonances can be required. The information on each individual resonance is enclosed in the XML subsection `<resonance>`, whose available parameters are summarised in Table 5. The resonance type is specified in `<particle>` according to its particle name as detailed in column 2 of Table 4. The tag `<external_legs>` defines the decay products of the resonance in terms of their positions in the list of `<outgoing>` (with counting starting from 3) of the subprocess pointing to this `<resonances id="XX">`. As long as the particle types of the decay products are consistent with the resonance, their position in the list of `<outgoing>` is not constrained, i.e. they are not forced to be adjacent. In the example in Listing 7, the top and antitop resonances define the requirement of the decay chain $t \to e^+\nu_e b$ and $\bar{t} \to \mu^-\bar{\nu}_\mu\bar{b}$, respectively, for the process $pp \to e^+\nu_e b\mu^-\bar{\nu}_\mu\bar{b}$.

parameter	possible values	default value
<particle>	t, t~, w+, w-, z, h	—
<external_legs>	list of integers (see text)	—
<phase_space_only>	true, false	true
<pole_approximation>	true, false	false
<polarisation_state>	-1, +1, 0, T, U	U

Table 5: Parameters of the <resonance> section of proc_card.xml and their default values.

By default, the presence of a resonances section tells MoCaNLO to construct only the integration channels that are compatible with the specified resonances, but it does not affect which contributions enter the matrix elements. This reduction of the amount of channels can speed up the convergence of the integration, because there are less channels to sample from. On the other hand, if there are significant contributions to the matrix elements in the considered fiducial region that are not sampled by the specified subset of channels, the convergence might be worse. For more information about the construction of integration channels we refer to Refs. [6–9].

Setting to false the additional tag <phasespace_only> (default true) in a specific <resonance> subsection causes the decay chain to also affect the matrix-element computation, namely only diagrams including that resonance will be part of the calculation. This is the starting point to define squared amplitudes including one or more on-shell resonances. Nevertheless, such a definition only based on a resonant-diagram selection breaks gauge invariance. This is why <phasespace_only> should only be set to false in combination with the <pole_approximation> tag (default false), where the matrix element is treated with a generic pole approximation algorithm, discussed in further detail in the following subsection.

5.2.1. Pole approximation: general features

The pole approximation [29] is based on the pole scheme [26–28], which provides a gauge-invariant way to separate resonant and non-resonant contributions at the level of the matrix element. When in MoCaNLO the XML resonance section includes the <phasespace_only> tag set to false in combination with <pole_approximation> set to true, the pole approximation is applied. Thus, only diagrams involving the specified resonances are included in the matrix elements. In addition, an on-shell-projected kinematics is used throughout the calculation except in the evaluation of resonances' propagators and when applying phase-space cuts, i.e. an event is still accepted or rejected according to its off-shell kinematics before on-shell projection. In Listing 8 two possible pole approximations are shown for the VBS process pp $\to e^+\nu_e\mu^-\bar{\nu}_\mu jj$, whose <incoming> and <outgoing> lists for a specific subprocess are given at the beginning of the same Listing.

In the block <resonances id="nested_hwm">, a pole approximation with a nested Higgs resonance is illustrated. There, two sets of external legs defining the Higgs and the W-boson resonances overlap. In order to deal with nested resonances within the pole approximation, MoCaNLO makes use of the general on-shell projection algorithm presented in Ref. [57]. The very same algorithm can also deal with single-pole approximations. A special treatment is only required for a few cases, which are separately discussed at the end of Sec. 5.2.2.

```
<subprocess id="sxs">
   <partonic_process>
      <incoming> s~ s                 </incoming>
      <outgoing> e+ ve mu- vm~ u u~ </outgoing>
      ...
   </partonic_process>
   ...
</subprocess>
...
<resonances id="nested_hwm">
   <resonance>
      <particle>            h       </particle>
      <external_legs>       3 4 5 6 </external_legs>
      <phasespace_only>     false   </phasespace_only>
      <pole_approximation> true     </pole_approximation>
   </resonance>
   <resonance>
      <particle>            w-      </particle>
      <external_legs>       5 6     </external_legs>
      <phasespace_only>     false   </phasespace_only>
      <pole_approximation> true     </pole_approximation>
   </resonance>
</resonances>
<resonances id="tpa_wpwmz">
   <resonance>
      <particle>            w+      </particle>
      <external_legs>       3 4     </external_legs>
      <phasespace_only>     false   </phasespace_only>
      <pole_approximation> true     </pole_approximation>
   </resonance>
   <resonance>
      <particle>            z       </particle>
      <external_legs>       5 6     </external_legs>
      <phasespace_only>     false   </phasespace_only>
      <pole_approximation> true     </pole_approximation>
   </resonance>
   <resonance>
      <particle>            z       </particle>
      <external_legs>       7 8     </external_legs>
      <phasespace_only>     false   </phasespace_only>
      <pole_approximation> true     </pole_approximation>
   </resonance>
</resonances>
```

Listing 8: Examples of pole approximations for the process $pp \to e^+\nu_e\mu^-\bar{\nu}_\mu jj$.

The behaviour of the on-shell projection can be tuned via the `<on_shell_projection id="...">` block in the `proc_card.xml`, which is linked in the subprocess definition in the `run_card.xml`. The elements of this block are summarised in Table 6. The on-shell projection implemented in MoCaNLO always preserves at least one invariant, which is by default the partonic CM energy. Using the tag `<preserve_system>`, the system whose total momentum is preserved can be changed from the default one, i.e. the partonic CM system (`cms`), to the one defined by the sum of the momenta of the on-shell-projected resonances (`cms_res`). Note that, when running a `real` contribution as part of an NLO calculation, the radiated particle is always included in the preserved system if `<preserve_system>` is set to `cms`. If instead `cms_res` is chosen, the radiation is included only for real corrections to the decays (`osp_type` > 0), while excluded for real corrections to the production (`osp_type` $= 0$). See discussion in Sec. 5.2.2.

Furthermore, the user may require to preserve additional invariants via the input tags of the sub-block `<preserve_invariants>`. When the tag `<search_invariants>` (default `0`) is set to 1, the code loops over all integration channels and selects the ones maximising the number of resonances that

parameter	possible values	default value
<preserve_system>	cms, cms_res	cms
<poldef_system>	cms, cms_res, lab	value of <preserve_system>
<preserve_directions>	list of integers (see text)	—
<keep_resonance_widths>	0, 1	0
<search_invariants>	0, 1	0
<subsystem>	full, res, nres, nres_tot, res+nres, res+nres_tot	full
<priority>	list of resonance identifiers (as in second column of Table 4)	h t* w* z
<fudge_factor>	0, 1	0
<fudge_factor_born>	0, 1	0
<fudge_width_range>	real number	3.0

Table 6: Parameters of the <on_shell_projection> section of proc_card.xml and their default values.

are not projected on shell. By default, Higgs bosons are given the highest priority, followed by top quarks, W bosons, and Z bosons. This behaviour can be modified using the input <priority> and providing a list of resonances as MoCaNLO identifiers (as defined in second column of Table 4). Different priorities can be required for particles and antiparticles of the same resonance using the appropriate identifiers, e.g. t/t~ and w+/w- for top quarks and W bosons, respectively. To give the same importance to particles and antiparticles, a star key can be used next to the resonance identifier, e.g. t* or w*. Resonances in the first entries of the list receive higher priority and integration channels with the largest number of propagators associated to them are selected. If no channel with at least one resonance of the <priority> list can be found, the code stops, since no invariant fulfilling the input criteria can be constructed. Conversely, if a subset of channels is found, the code further filters them according to its default priority criteria. Additionally, more importance is given to those channels that involve resonances decaying into a smaller amount of particles. Once an integration channel is finally selected, invariants are collected from all resonant propagators that are not set on shell. Invariants corresponding to the partonic CM energy are excluded. Moreover, since the algorithm tries to search for invariants that can be preserved both at LO and NLO, invariants including momenta from all final-state photons and/or gluons are also discarded. The final selection of preserved invariants is printed to the standard output of the run.

An additional filter on the invariants to be preserved can be activated via the flag <subsystem>. By default (full), all invariants are kept. By choosing <subsystem> = res, only invariants in the sub-system defined by the decay products of the on-shell-projected resonances are kept, while the value nres restricts the momenta of the invariants to the complementary system, i.e. all final-state momenta that are not part of on-shell-projected resonances. Invariants from both sets are chosen with the value res+nres. Moreover, using nres_tot and res+nres_tot allows to preserve the invariant of the complementary system as a whole. This latter option can be used for instance to address the invariant mass of the tag jets in the context of VBS (see Ref. [57]). Note that the set of additionally preserved invariants can only be a subset of the system defined by <preserve_system>,

```
<on_shell_projection id="nested_hwm">
   <preserve_system>       cms  </preserve_system>
   <keep_resonance_widths> 1    </keep_resonance_widths>
   <preserve_invariants>
      <search_invariants>  1    </search_invariants>
      <subsystem>          full </subsystem>
      <priority>           w* z </priority>
   </preserve_invariants>
</on_shell_projection>
```

Listing 9: Additional on-shell projection information in `proc_card.xml` extending Listing 8.

meaning that when the latter is set to `cms_res`, `<subsystem>` can only be set to `res`, otherwise the code stops.

An example of an `<on_shell_projection id="...">` block is shown in Listing 9. There, the conservation of the CM energy is required explicitly using the `<preserve_system>` tag. In the block `<preserve_invariants>`, additional invariants to be preserved are searched in the full final state, giving higher priority to the ones that can be associated to intermediate W bosons (regardless of the charge) and, with lower priority, to intermediate Z bosons.

In some applications one may want to preserve the directions of a subset of final-state particles. By default, the on-shell projection described in Ref. [57] preserves all directions of the decay products of a resonance in the rest frame of the latter, but not in the laboratory frame. This behaviour can be modified via the optional `<preserve_directions>` tag in the `<on_shell_projection>` section. The latter must contain a list of integers referring to the position of the final-state particles whose off-shell directions in the laboratory frame one wants to preserve. This feature, which allows to reproduce the on-shell projection introduced in Ref. [29], is only available for resonances decaying into two particles, where no more than one decay-product direction per resonance can be preserved, i.e. each integer in the `<preserve_directions>` list must belong to a different resonance. Moreover, this option cannot be used for a `real` run type.

As part of the pole-approximation algorithm, the widths of the on-shell-projected resonances must be set to zero everywhere in the calculation except in the propagators of the specified resonances. Nevertheless, when a pole approximation with n resonances is performed, a process may have a larger number of resonances than the specified ones with the same types of particles. This is the case for our example subprocess `<subprocess id="sxs">`, which also admits triply-resonant contributions, as evidenced by the second pole approximation `<resonances id="tpa_wpwmz">` in Listing 8. When this happens, setting the decay widths of the resonances to zero causes the phase-space integration to run into singularities, unless they are excluded by cuts. Even if no general solution to the problem exists, MoCaNLO offers two ways to handle divergent resonant propagators in the context of the pole approximation.

A simple workaround is to keep the resonance widths non-zero [65]. This can be enforced by switching on the tag `<keep_resonance_widths>` in the `<on_shell_projection>` section (see Listing 9). If not set, the code works as usual and sets to zero the decay widths in the pole-approximated squared amplitudes except in the propagators of the required resonances.

On the usage of `keep_resonance_widths`

- The flag `<keep_resonance_widths>` must not be used for pole-approximated calculations with radiating charged resonances at NLO (see Sec. 5.2.2). Keeping a non-zero width at NLO would lead to inconsistencies when charged resonances are projected on shell, since singularities stemming from the associated propagators in the real squared amplitude are already accounted for by dedicated subtraction counterterms [65].
- Keeping a non-zero width for the resonances cures singularities in the phase-space integration but introduces a gauge dependence of the squared amplitude within the pole approximation. Even if in most cases this effect is expected to be negligible, its impact on final results should be verified on a process-by-process basis.
- This option may change the value of the EW coupling, if the latter is computed using complex masses for the W and Z bosons as inputs of the calculation (see option `<masses_in_alpha_gf>` in Sec. 6.1).

If a process in the pole approximation is protected at LO against singularities from extra resonant propagators by the definition of the fiducial phase space, so that problems can only appear at NLO, MoCaNLO offers an alternative regularisation. Its usage is controlled via the block `<fudge_factor_regularisation>`. Within this block, if `<fudge_factor>` is set to `1`, squared amplitudes for the `real` and `idip` contributions receive correction factors that regularise all resonant propagators that appear in a given partonic process. The propagators are found by the code by looping over all integration channels and collecting all resonant propagators that are not projected on shell. For each propagator i a factor $\mathcal{F}_i$ is constructed,

$$\mathcal{F}_i(s_i) = \begin{cases} 1 & \text{if} \quad |\sqrt{s_i} - M_i| > c_{\mathcal{F}}\,\Gamma_i \\ \frac{(s_i - M_i^2)^2}{(s_i - M_i^2)^2 + \Gamma_i^2 M_i^2} & \text{if} \quad |\sqrt{s_i} - M_i| < c_{\mathcal{F}}\,\Gamma_i \end{cases}, \tag{2}$$

where s_i refers to the invariant mass of the unregularised resonance i having a pole mass M_i and a width Γ_i. The full fudge factor results from the products of the $\mathcal{F}_i$ for all propagators found by the algorithm. In the previous formula, the parameter $c_{\mathcal{F}}$ restricts the range of application of the fudge factor. Its value can be controlled by the input `<fudge_width_range>` (default value `3.0`). Finally, even if the fudge-factor regularisation is mostly useful to deal with resonance singularities appearing at NLO, the usage of the fudge factor to regularise LO processes can also be enforced by setting `<fudge_factor_born>` to `1` (when also `<fudge_factor>` has been set to `1`).

A block `<fudge_factor_regularisation>` is illustrated in Listing 10. This example must be read together with Listing 11, which defines the pole approximation for the process $pp \to e^+\nu_e\mu^-\bar{\nu}_\mu jj$. In the context of an NLO calculation (see Sec. 5.2.2), the section `<on_shell_projection>` of Listing 10 regularises the resonant propagators other than the ones projected on shell.

As a final observation, it is worth noting that, while the use of pole approximations is supported along with the final-state merging of different partonic processes, all merged partonic processes must have the same resonance structure. If this is not the case, the code will still compute the correct results but with a very low convergence rate.

```
<on_shell_projection id="wpwm_reg">
   <preserve_system>       cms_res  </preserve_system>
   <fudge_factor_regularisation>
      <fudge_factor>       1        </fudge_factor>
      <fudge_width_range>  4.0      </fudge_width_range>
   </fudge_factor_regularisation>
</on_shell_projection>
```

Listing 10: Additional on-shell projection information in `proc_card.xml` for the pole approximation defined in Listing 11.

5.2.2. Pole approximation at NLO

While MoCaNLO can deal with resonances in the pole approximation in full generality at LO, only a restricted set of processes is currently supported in pole approximation at NLO accuracy. Limitations essentially come from the treatment of the soft and collinear singularities in the real-radiation contributions, which becomes more cumbersome compared to the full off-shell description.

In particular, a fully-fledged NLO (QCD or EW) calculation in the pole approximation is only possible under the following conditions:

- The process involves *one or two weak bosons* as intermediate states, treated in the single- or double-pole approximation, respectively. Neither processes with three or more resonances, nor those with a nested resonance structure can be handled at NLO. Top quarks cannot be treated either in this regard.
- The resonances undergo *two-body leptonic decays* at LO, and the real-radiation contributions lead to resonances that decay at most into three particles, i.e. *up to three-body decays in real corrections* at NLO. Hadronic decays of resonances are not supported.
- Additional limitations arise for *charged resonances*. Since an intermediate charged resonance can also radiate, dedicated counterterms are required for a complete subtraction of singularities, when such a resonance is projected on shell (for more details see Ref. [65]). MoCaNLO only supports processes at NLO with leptonically-decaying W bosons. Colour-charged resonances like top quarks cannot be handled at NLO within the pole approximation.
- As a final remark, a full treatment of *non-factorisable corrections* [31, 32] is not supported. Even if both factorisable and non-factorisable corrections can be computed for the virtual contribution (as discussed in Sec. 5.2.3), only factorisable ones are available for the real contribution. In any case, the impact of these corrections is known to be small if both the real and virtual contributions are treated in the pole approximation, since in their sum non-factorisable corrections cancel in observables that are inclusive with respect to the decay products of the resonances [30, 74–77].

An NLO calculation in the pole approximation for a given subprocess can be performed by running all relevant contributions (`born`, `virt`, `idip`, and `real`) with the linking tag `<resonances id="XX">` pointing to the same XML resonances block in the `proc_card.xml`, which selects a particular pole approximation, as explained in Sec. 5.2.1. Additionally, for the real part one needs to compute separate contributions where the radiation is emitted from the production and the decay of the N on-shell-projected resonances. This means that one is required to submit $N + 1$ separate real runs for a given subprocess to obtain a complete NLO pole-approximated result. Note that the radiated particle for the `real` in the pole approximation is identified by comparing the `<partonic_process>`

```
<runs directory="vbs_wpwm">
   <run id="1">
      <type>             real          </type>
      <subdirectory>     ud/real       </subdirectory>
      <subprocess        id="ud"       />
      ...
      <osp_type> 0 <osp_type/>
      <resonances           id="wpwm" />
      <on_shell_projection id="wpwm" />
   </run>
   <run id="2">
      <type>             real          </type>
      <subdirectory>     ud/real       </subdirectory>
      <subprocess        id="ud"       />
      ...
      <osp_type> 1 <osp_type/>
      <resonances           id="wpwm" />
      <on_shell_projection id="wpwm" />
   </run>
   <run id="3">
      <type>             real          </type>
      <subdirectory>     ud/real       </subdirectory>
      <subprocess        id="ud"       />
      ...
      <osp_type> 2 <osp_type/>
      <resonances           id="wpwm" />
      <on_shell_projection id="wpwm" />
   </run>
   ...
</runs>
```

Listing 11: Example of real contributions in pole approximation in the `run_card.xml`.

and the `<real_process>`: This is sufficient for the cases supported by MoCaNLO, where a real configuration only admits one underlying born. The information on the origin of the radiation is instead provided directly in the `run_card.xml` as part of the subprocess definition, as illustrated in Listing 11. This is done using the tag `<osp_type>` (default value `0`), which assumes the value `0` for the production contribution, or any number from `1` to `N` for the decay contributions of the N resonances. These numbers refer to the ordering in which resonances are listed in the shared `<resonances id="...">` block.

A possible `proc_card.xml` for the real contributions in Listing 11 is shown in Listing 12, where a double-pole approximation for the process $pp \to e^+\nu_e\mu^-\bar{\nu}_\mu jj$ is considered. Note that the position of the radiation in the `<outgoing>` list of `<real_process>` (a photon in our example) does not need to be contiguous to the decay products of the on-shell-projected resonances (the leptons in our example). For illustrative purposes, the example in Listing 12 requires to preserve the total momentum of the two W bosons that are projected on shell via the `<preserve_system>` tag. This choice is usually more natural than the default one for specific applications, for example when studying vector-boson polarisations (see Sec. 5.2.4).

Further remarks on the single-pole approximation The general pole-approximation algorithm implemented in MoCaNLO [57] covers also calculations in the single-pole approximation, up to special treatments needed for a few cases.

- Two-to-two process: at variance with the general case, the on-shell projection rescales initial-state momenta and (when computing NLO corrections) the final-state momentum of the radiation, if this is not part of the decay products of the resonance of interest. The rescaling

```
<subprocess id="ud">
   <partonic_process>
      <incoming> u d                  </incoming>
      <outgoing> e+ ve mu- vm~ u d    </outgoing>
      ...
   </partonic_process>
   <real_process>
      <incoming> u d                  </incoming>
      <outgoing> e+ ve mu- vm~ u d a  </outgoing>
      ...
   </real_process>
</subprocess>
<on_shell_projection id="wpwm">
   <preserve_system>    cms_res   </preserve_system>
</on_shell_projection>
<resonances id="wpwm">
   <resonance>
      <particle>            w+     </particle>
      <external_legs>       3 4    </external_legs>
      <phasespace_only>     false  </phasespace_only>
      <pole_approximation>  true   </pole_approximation>
   </resonance>
   <resonance>
      <particle>            w-     </particle>
      <external_legs>       5 6    </external_legs>
      <phasespace_only>     false  </phasespace_only>
      <pole_approximation>  true   </pole_approximation>
   </resonance>
</resonances>
```

Listing 12: Examples of pole approximations for the process $pp \to e^+\nu_e\mu^-\bar{\nu}_\mu jj$.

```
<run id="1">
   <type>             vnfc  </type>
   <subdirectory> uxu/vnfc  </subdirectory>
   <subprocess        id="uxu"/>
   <run_parameters    id="eex_nlo"/>
   <model_parameters  id="eex_nlo"/>
   <cuts              id="eex"/>
   <recombinations    id="eex"/>
   <resonances        id="w_psp"/>
   <histograms        id="none"/>
</run>
```

Listing 13: Run definition in `run_card.xml`.

factor is equal to the ratio of the pole mass over the invariant mass of the resonance. Since in this case the on-shell projection has to modify the incoming momenta, it cannot even preserve the partonic CM energy. Therefore, the input provided via the tag `<preserve_system>` is ignored.

- Single resonance with at least one additional massive external particle: the pole approximation proceeds as in the general case, but when evaluating a real contribution to the production, i.e. `osp_type` = `0`, the momentum of a bremsstrahlung particle (a photon or a gluon) is never included in the preserved system (in contrast to the default behaviour, if `cms` was chosen), i.e. the momentum of the radiation is never modified. For real corrections to the decays, no change in the algorithm is needed instead.

Note that the pole approximation for a single intermediate charged resonance produced in association with massless particles is ill defined close to threshold, unless the resonance's invariant mass

is constrained by kinematic cuts preventing it from becoming arbitrarily close to the pole mass. Consequently, processes with a single W boson or a top quark in association with massless particles are not supported by MoCaNLO, while for instance cases with a single Z boson decaying into charged-fermion pairs are supported and treated as in the general case.

5.2.3. Virtual non-factorisable corrections

Within the restrictions discussed in Sec. 5.2.2, MoCaNLO allows to compute all contributions (i.e. `born`, `virt`, `idip`, and `real`) in the pole approximation, including only factorisable corrections for the virtual and real terms. This provides a consistent NLO result in this approximation and is also the only way to define a polarised cross section, as described in Sec. 5.2.4. However, the pole approximation can also be used to speed up the computation of virtual corrections, whose evaluation is particularly expensive. A solution, which was adopted for instance in Refs. [29, 31], is to compute all contributions entering an NLO calculation exactly, with the exception of the virtual one, where a pole approximation was used. This requires in MoCaNLO to run for each subprocess a `virt` and a `vfnc` run type both specifying a `<resonances id="XX">` section in the `run_card.xml`. If both factorisable and non-factorisable virtual corrections are accounted for, their IR singularities must be cancelled applying the on-shell projection to the terms containing the I operator in the integrated dipole contribution in the same way. The P- and K-operator terms, on the other hand, must be evaluated with the off-shell kinematics like the real corrections. This introduces a mismatch that is of the order of the intrinsic error of the pole approximation. A proper evaluation of the I operator, in line with our discussion, can be achieved in MoCaNLO by replacing each `idip` contribution for a specific subprocess by an `idpi` and an `idpk` run pointing to the same subprocess. The `idpi` runs should specify a `<resonances id="XX">` section with `pole_approximation=true`, while the `idpk` ones must be run with off-shell kinematics. Examples of how to perform these kind of computations are provided in the folder `validated_processes`, in particular the processes in the double-pole approximation in the `ttbar/pp_tt` folder. It is worth noting that while the implementation of the non-factorisable corrections closely follows Ref. [30–32] and is therefore very general (arbitrary number of non-nested resonances), it has only been validated for double-pole approximations involving two identical resonant particles [34, 35, 38, 49].

Warning

The computation of the non-factorisable virtual corrections is limited to processes without nested resonances. Moreover, processes with a single charged resonance in association with massless particles in the final state are also not allowed, since the pole approximation is ill defined in these cases.

5.2.4. Selection of polarisations

MoCaNLO can compute cross sections for internal polarised vector bosons at NLO accuracy. The definition of polarised cross sections is based on the pole approximation, as described for instance in Ref. [59]. As a consequence, polarised results are only available for processes that can be calculated in this approximation (see discussion at the beginning of Sec. 5.2.2).

<polarization_state>	Description
-1 / +1	Left- and right-handed
0	Longitudinal
T	Transverse
U	Unpolarised (default)

Table 7: Available polarisation states.

```
<on_shell_projection id="ww_osp">
   <poldef_system> cms_res  </poldef_system>
</on_shell_projection>
<resonances id="ww_psp">
   <resonance>
      <particle>           w+    </particle>
      <external_legs>      3 4   </external_legs>
      <phasespace_only>    false </phasespace_only>
      <pole_approximation> true  </pole_approximation>
      <polarization_state> T     </polarization_state>
   </resonance>
   <resonance>
      <particle>           w-    </particle>
      <external_legs>      5 6   </external_legs>
      <phasespace_only>    false </phasespace_only>
      <pole_approximation> true  </pole_approximation>
      <polarization_state> 0     </polarization_state>
   </resonance>
</resonances>
```

Listing 14: Examples of pole approximations for the process $pp \to e^+\nu_e\mu^-\bar{\nu}_\mu jj$.

To run simulations for polarised vector bosons, one must add to each resonance block, where the pole approximation has been enabled, the required polarisation state via the tag <polarization_state>. Allowed values are reported in Table 7. In the example in Listing 14, two on-shell W bosons having transverse and longitudinal polarisations are required.

The definition of polarisation states is Lorentz-frame dependent, and consequently the result for the polarised cross sections too. By default, MoCaNLO defines polarisations in the same frame as requested in <preserve_system> (see Sec. 5.2.1). Nonetheless, a different reference frame for the polarised states can be set in the XML section <on_shell_projection id="..."> using the <poldef_system> tag. In Listing 14, the pole approximation preserves the partonic CM energy of the process (default when <preserve_system> is not specified), but computes polarisations in the CM frame of the two vector bosons. The possible input values for <poldef_system> (see Table 6) are those of <preserve_system> and the value lab, which allows to define polarisations in the laboratory frame. Note that this is the default frame used for the definition of polarisations in case of a single-pole approximation.

6. Specifying the simulation parameters: the param_card.xml

The parameter card param_card.xml contains two types of sections. These are labelled <model_parameters id="...">, where parameters of the particle model are given, and <run_parameters id="...">, which specifies parameters for the MC run. The contents of these two section types are discussed in the following.

```
<model_parameters id="ttx_nlo">
    <top_mass>                  172.0              </top_mass>
    <top_width>                   1.31670          </top_width>
    <z0_mass>                    91.1876           </z0_mass>
    <z0_width>                    2.5096596343     </z0_width>
    <w_mass>                     80.399            </w_mass>
    <w_width>                     2.0997360974     </w_width>
    <higgs_mass>                120.0              </higgs_mass>
    <higgs_width>                 0.0              </higgs_width>
    <scheme_alpha>                GF               </scheme_alpha>
    <masses_in_alpha_gf>         -1                </masses_in_alpha_gf>
    <alphagf>                     7.555786d-03     </alphagf>
    <renormalization_scale>     172.0              </renormalization_scale>
    <factorization_scale>       172.0              </factorization_scale>
    <infrared_scale>            172.0              </infrared_scale>
    <neutrino_species_tagging>    1                </neutrino_species_tagging>
    <redefine_jet_types>
        <jet_particle>          b b~               </jet_particle>
        <jet_type>              2 27               </jet_type>
    </redefine_jet_types>
</model_parameters>
```

Listing 15: Model parameters in `param_card.xml`.

A single `param_card.xml` can contain several subsections of the same type, each with a unique `id` attribute that is used for linking in the `run_card.xml` (see Sec. 4). This can be used to include different parameter sets, e.g. for LO and NLO calculations. The exemplary Listing 15 contains the NLO value of the top-quark width Γ_t for the NLO calculation of the process $pp \to e^+\nu_e b \mu^- \bar{\nu}_\mu \bar{b}$ and is thus denoted with the attribute `id="ttx_nlo"`. An additional section `<model_parameters id="ttx_lo">` containing the LO value of Γ_t could be present in `param_card.xml`. Similarly, different sections `<run_parameters id="ttx_nlo">` and `<run_parameters id="ttx_lo">` could point to the usage of NLO and LO PDF sets, respectively.

6.1. Model parameters

All parameters available in the `<model_parameters>` section of `param_card.xml` are summarised in Tables 8 and 9 together with their default values, and Listing 15 shows an example of their use in `param_card.xml`.

Masses and widths of particles in GeV are set via `<particle_mass>` and `<particle_width>`, respectively. The electron and muon masses are only used as regulators of initial-state singularities for incoming electrons/positrons or muons and are considered as massless otherwise. The four light quarks are considered as massless throughout. While the masses of the bottom quark and the τ lepton are zero by default, finite masses are allowed as long as these particles do not appear as radiating particles in NLO calculations.

MoCaNLO uses consistently pole masses M and widths Γ, which are related to the complex poles μ via

$$\mu^2 = M^2 - \mathrm{i}M\Gamma.$$

parameter	XML element	default value
m_{t}	<top_mass>	172.0
Γ_{t}	<top_width>	1.4
m_{b}	<bottom_mass>	0.0
Γ_{b}	<bottom_width>	0.0
m_{τ}	<tau_mass>	0.0
m_{μ}	<muon_mass>	0.1056584
m_{e}	<electron_mass>	$0.51099895 \times 10^{-3}$
$M_{\mathrm{Z}}^{\mathrm{OS}}$	<ons_z0_mass>	91.1880
$\Gamma_{\mathrm{Z}}^{\mathrm{OS}}$	<ons_z0_width>	2.4955
$M_{\mathrm{W}}^{\mathrm{OS}}$	<ons_w_mass>	80.3692
$\Gamma_{\mathrm{W}}^{\mathrm{OS}}$	<ons_w_width>	2.085
M_{Z}	<z0_mass>	calculated
Γ_{Z}	<z0_width>	calculated
M_{W}	<w_mass>	calculated
Γ_{W}	<w_width>	calculated
M_{H}	<higgs_mass>	125.0
Γ_{H}	<higgs_width>	4.07×10^{-3}
G_{F}	<fermi_constant>	1.166390×10^{-5}
–	<masses_in_alpha_gf>	0
–	<scheme_alpha>	gf
$\alpha_{G_{\mu}}$	<alphagf>	$7.555310522369 \times 10^{-3}$
$\alpha(0)$	<alpha0>	7.2973525×10^{-3}
$\alpha(M_{\mathrm{Z}}^2)$	<alphaz>	7.7983287×10^{-3}
$\alpha_{\overline{\mathrm{MS}}}(Q_0^2)$	<alpha_starting_value>	7.8609176×10^{-3}
Q_0	<alpha_starting_scale>	91.188
α_{s}	<alphas>	none
–	<alphas_running>	1 if hadrons in initial state, 2 otherwise
$\alpha_{\mathrm{s}}(Q_0^2)$	<alphas_starting_value>	0.118
Q_0	<alphas_starting_scale>	91.188
–	<n_loops>	2
μ_{R}	<renormalization_scale>	μ_{F} or M_{Z}
μ_{F}	<factorization_scale>	μ_{R} or M_{Z}
μ_{IR}	<infrared_scale>	100.0
–	<neutrino_species_tagging>	0
–	<redefine_jet_types>	—

Table 8: Elements of the <model_parameters> section of param_card.xml and their defaults.

parameter	XML element	default value
m_{reg}	<massless_mapping_mass>	10^{-6}
p	<massless_mapping_power>	0.9
q	<subtraction_mapping_power>	0.8

Table 9: Additional technical parameters of the <model_parameters> section of param_card.xml and their defaults.

Since for the weak vector bosons usually the on-shell masses M^{OS} and widths Γ^{OS} are quoted, these can be given alternatively using the input tags <ons_z0_mass>, <ons_z0_width>, <ons_w_mass>, and <ons_w_width>. From those, the pole masses and widths are calculated via [78]

$$M = \frac{M^{\text{OS}}}{\sqrt{1+(\Gamma^{\text{OS}}/M^{\text{OS}})^2}} \qquad \text{and} \qquad \Gamma = \frac{\Gamma^{\text{OS}}}{\sqrt{1+(\Gamma^{\text{OS}}/M^{\text{OS}})^2}}.$$

If no input values are provided, MoCaNLO determines the pole masses from default on-shell masses. Therefore, for weak vector bosons, pole masses and widths are always recomputed by MoCaNLO, unless directly specified via input. Note that pole and on-shell values can not be provided together in the param_card.xml. If so, the code stops, asking to correct the input.

In MoCaNLO, by default the G_{F} scheme is used for the renormalisation of the EW coupling α. For <masses_in_alpha_gf> $= -1$, the input value <alphagf> is used to set α. Otherwise, two cases are possible, where the value of α is derived from the value of G_{F}, set via <fermi_constant>, and from the masses of the W and Z bosons. For <masses_in_alpha_gf> $= 0$ (default), α is calculated with real masses via

$$\alpha = \frac{\sqrt{2}G_\mu}{\pi} \operatorname{Re} M_{\text{W}}^2 \left(1 - \frac{\operatorname{Re} M_{\text{W}}^2}{\operatorname{Re} M_{\text{Z}}^2}\right),$$

while for <masses_in_alpha_gf> $= 1$ complex masses are used according to

$$\alpha = \frac{\sqrt{2}G_\mu}{\pi} \left| M_{\text{W}}^2 \left(1 - \frac{M_{\text{W}}^2}{M_{\text{Z}}^2}\right)\right|.$$

Additional input schemes for α are available and can be chosen using the input tag <scheme_alpha>. For <scheme_alpha> $= 1$ or <scheme_alpha> $=$ gf the G_μ scheme is selected as discussed above. For <scheme_alpha> $= 2$ or <scheme_alpha> $=$ alpha0, the input value <alpha0> is used to set α, while for <scheme_alpha> $= 3$ or <scheme_alpha> $=$ alphaz, the input value <alphaz> is employed. Finally, for <scheme_alpha> $= 4$ or <scheme_alpha> $=$ alphamsbar, the $\overline{\text{MS}}$ scheme is chosen with α set to <alpha_starting_value> and the corresponding scale set to <alpha_starting_scale>. The choice of <scheme_alpha> fixes the input value of α and its renormalisation. For instance, in the $\overline{\text{MS}}$ scheme the counterterm in Eq. (427) of Ref. [26] is used. More details on the renormalisation of α can be found in the Recola manual [11]. Note that the use of differently renormalised couplings for external photons is steered via the parameters <tree_e0_order> in proc_card.xml.

In MoCaNLO, the value of the strong coupling constant α_{s} can be either provided by the user or calculated using its running. A fixed value of α_{s} can be set using the tag <alphas>. The corresponding renormalisation scale μ_{R} is then assumed to be fixed and must be set via the tag <renormalization_scale>. Even when using a fixed α_{s} value, a running of this coupling

is required if a scale variation is to be performed (see Sec. 6.2). For this reason, the value of <renormalization_scale> corresponding to the given value <alphas> must be specified.

In general, MoCaNLO provides two options for the running of α_s, controlled by the run parameter <alphas_running> (see Sec. 6.2). If <alphas_running> is set to 1, the running provided by LHAPDF is selected. If <alphas_running> is set to 2, the running provided by Recola is instead used. In the latter case, a starting scale Q_0 and the value of the coupling $\alpha_s(Q_0^2)$ at that scale are required and can be specified via the parameters <alphas_starting_scale> and <alphas_starting_value>. The running of the coupling also depends on the loop order of the running, fixed via the parameter <n_loops>, which can be 1 or 2. Note that MoCaNLO uses the five-flavour scheme.

As noted before, the running of the coupling is required even for a fixed scale if a scale variation is performed. Therefore, also in this case a value has to be given for <alphas_running>. The default value of <alphas_running> depends on the parameters <beam1> and <beam2> (see Sec. 6.2). If at least one of the beams is a proton beam, <alphas_running> defaults to 1, otherwise it is set to 2.

If fixed renormalisation and factorisation scales μ_R and μ_F are employed, their values must be passed using the tags <renormalization_scale> and <factorization_scale>. If only one of the two scales is set by the user, MoCaNLO will automatically set both to the given value. If none of the two is set, they will both be assigned the value M_Z. The choice of a fixed or dynamical scale is determined by the run parameter <dynamical_scale_type> (see Sec. 6.2). Note that the IR scale μ_{IR} used for the computation of virtual corrections and integrated dipoles is not assumed to be equal to any of the previously defined scales in MoCaNLO and can be independently set via the <infrared_scale> input. Since the result must not depend on μ_{IR}, varying this parameter offers a useful consistency check on the calculation.

The technical parameters m_{reg}, p, and q enter the mappings of massless propagators and singularities in the subtraction terms in the MC integration. Their values can be adjusted via the tags <massless_mapping_mass>, <massless_mapping_power>, and <subtraction_mapping_power>, respectively. These influence the integration performance and may be tuned, but m_{reg} should be smaller than the lowest physical scale of the process by several orders of magnitude, and $p, q \lesssim 1$ should hold.

Setting <neutrino_species_tagging> to 1 allows to switch on different jet types for neutrino species (see Table 4 in Sec. 5.1), which can be useful for tagging or producing histograms based on specific neutrino truth momenta. If this parameter is set to 0, all neutrinos have the same jet type (6).

The subsection <redefine_jet_types> allows to redefine the default <jet_type> of a subset of particles. The jet types in the list <jet_type> ... </jet_type> become the jet types of the corresponding particles in the list <jet_particle> ... </jet_particle>. In the example in Listing 15, the jet type of the antibottom quark is set to 27, which allows to recombine and tag them independently of the bottom quark and to plot corresponding observables. As a further example, the jet types of bottom and antibottom quarks or of the photon can be set to 1 if these shall be treated as light jets.

6.2. Run parameters

The section <run_parameters> of param_card.xml defines several parameters controlling the MC run. These parameters together with their possible input values are listed in Tables 10 and 11. In Listing 16, run parameters for the NLO calculation of $pp \to e^+\nu_e b\mu^-\bar{\nu}_\mu \bar{b}$ are displayed as an example.

parameter	possible values	default value
<energy>		(group)
<cms_beam_energy>	positive real number	none
<energy_beam1>	positive real number	none
<energy_beam2>	positive real number	none
<scale>		(group)
<dynamical_scale_type>	$0, 1$	0
<scale_factors_f>	list of real numbers	1.0
<scale_factors_r>	list of real numbers	1.0
<n_scales_for_histograms>	integer	maximum # of scale factors
<pdfs>		(group)
<beam1>	p,e+,e-,mu+,mu-	p
<beam2>	p,e+,e-,mu+,mu-	p
<pdf_set>	see text	—
<pdf_set2>	see text	—
<alphas_running>	$1, 2$	see text
<pdf_scheme>	string	see text
<pdf_scheme2>	string	see text
<pdf_starting_scale>	positive real number	0.511×10^{-3}
<helicity_em>	$-1, 0, 1$	0
<helicity_ep>	$-1, 0, 1$	0
<dipoles>		(group)
<alpha_fs_fs>	real $\in (0, 1]$	1.0
<alpha_fs_is>	real $\in (0, 1]$	1.0
<alpha_is_fs>	real $\in (0, 1]$	1.0
<alpha_is_is>	real $\in (0, 1]$	1.0
<DIS_scheme>	$0, 1$	0
<integration_termination>		(group)
<n_target_accepted_events>	integer ≥ 1000	10^{15}
<max_generated_events>	integer	10^{15}
<max_vanishing_events>	integer	100000
<target_relative_precision>	integer	3
<target_absolute_precision>	integer	12
<time_wall>	days-hours:minutes	—

Table 10: Parameters of the <run_parameters> section of param_card.xml that fix the kind of calculation and their defaults.

```
<run_parameters id="ttx_nlo">
    <output_level>     4      </output_level>
    <cms_beam_energy> 8000   </cms_beam_energy>
    <scale>
        <dynamical_scale_type> 1        </dynamical_scale_type>
        <scale_factors_f>      1 0.5 2 </scale_factors_f>
    </scale>
    <pdfs>
        <pdf_set>           MSTW2008lo90cl </pdf_set>
        <pdf_mapping_emin> 120d0           </pdf_mapping_emin>
    </pdfs>
    <integration_termination>
        <n_target_accepted_events>  1 000 000 000 </n_target_accepted_events>
        <target_relative_precision> 5             </target_relative_precision>
        <target_absolute_precision> 5             </target_absolute_precision>
        <time_wall>                 1-07:30       </time_wall>
    </integration_termination>
    <dipoles>
        <alpha_fs_fs> 1d-2 </alpha_fs_fs>
        <alpha_fs_is> 1d-2 </alpha_fs_is>
        <alpha_is_fs> 1d-2 </alpha_is_fs>
        <alpha_is_is> 1d-2 </alpha_is_is>
    </dipoles>
</run_parameters>
```

Listing 16: MC run parameters in `param_card.xml`.

The run parameters can be grouped in `param_card.xml` into the subsections labelled `<energy>`, `<scale>`, `<pdfs>`, `<dipoles>`, `<integration_termination>`, `<output>`, `<intermediate_output>`, `<phase_space_generator>`, and `<running_options>`, but these tags are ignored by MoCaNLO.

Table 10 summarises the parameters that are needed to define the run.

<energy> The collider CMS energy is defined in `<cms_beam_energy>` in GeV. If this parameter is given, this energy is assumed to be equally distributed between both beams. For asymmetric colliders, the energies of the beams can be defined with `<energy_beam1>` and `<energy_beam2>` in GeV.

<scale> This subsection contains parameters related to the renormalisation and factorisation scales μ_R and μ_F. The parameter `<dynamical_scale_type>` fixes the type of scales to be used. For `<dynamical_scale_type>` $= 0$, fixed scales are chosen, whose values are passed via `<renormalization_scale>` and `<factorization_scale>` in the section `<model_parameters>`. Dynamical scales can be chosen by setting `<dynamical_scale_type>` $= 1$. The definition of the scales itself must be specified in the file `cut_card.xml` (see Sec. 7.3 for details).

The scales μ_F and μ_R can be independently re-scaled to perform an estimation of the scale uncertainty. The scaling factors are passed as lists of space-separated real numbers via the tags `<scale_factors_f>` and `<scale_factors_r>` for μ_F and μ_R, respectively. The matrix elements and the PDFs are computed using the values of μ_F and μ_R as specified above times the first entry of `<scale_factors_f>` and `<scale_factors_r>`, respectively. The first entry of each list corresponds to the central scale. Note that the runtime does not grow proportional to the number of scale factors. If the tags `<scale_factors_f>` and `<scale_factors_r>` are not provided, a single scaling factor equal to 1 is used for both scales. If only one list of factors is provided or if the lists have different lengths, the shorter list is extended to the length of the longer list using additional entries equal to 1. When a scale variation is performed, the results of each scale factor are written to different directories (see Sec. 3), whose names correspond to the position of the factors in the lists `<scale_factors_f>` and `<scale_factors_r>`.

The integer <n_scales_for_histograms> defines for how many scale factors the histograms are computed at most (see Sec. 8), and its default value is the larger of the number of scales given in the lists <scale_factors_f> and <scale_factors_r>. Histograms are produced for the first <n_scales_for_histograms> factors in the lists <scale_factors_f> and <scale_factors_r>. If <n_scales_for_histograms> is larger than the length of the lists, the run stops.

<pdfs> This subsection contains parameters related to the incoming particles and the employed PDFs. The type of the beams is fixed via the input tags <beam1> and <beam2>, which can take the values p, e+, e−, mu+ or mu−. By default, proton (p) beams are assumed.

The PDF set is chosen via the tag <pdf_set>. For proton–proton collisions, the LHAPDF library is used (see Sec. 2). Different PDF sets for the second beam can be used by setting <pdf_set2> to a value different from <pdf_set>. This way, proton–lepton collisions can be simulated. If <pdf_set2> is not given, it is assumed to be equal to <pdf_set>.

Warning

MoCaNLO has not been thoroughly tested for proton–lepton collisions.

The tag <pdf_scheme> fixes the subtraction scheme for the IR singularities in the collinear counterterms and should match the IR subtraction of the PDFs. Different schemes imply different finite contributions in the collinear counterterms. By default <pdf_scheme> is set to MSbar, which is the only available setting for proton PDFs in MoCaNLO. Different settings are instead available for lepton PDFs, as discussed below.

The strong coupling constant can either be taken from the PDFs or be calculated by Recola: This behaviour is controlled by the parameter <alphas_running> in the <model_parameters> of the param_card.xml (see Sec. 6.1). If <alphas_running> is set to 1, α_s is obtained from the PDF set, if it is set to 2, α_s is calculated by Recola. The default value of <alphas_running> depends on the parameters <beam1> and <beam2>. If at least one of the beams is a proton beam, <alphas_running> defaults to 1, otherwise it is set to 2.

To simulate lepton–lepton collisions, the tag <pdf_set> can be set to none. In this case, no PDFs are used and the initial-state collinear singularities are regularised with the electron or muon masses (see Sec. 6.1). Alternatively, MoCaNLO provides an internal implementation of LO lepton PDFs according to Appendix A of Ref. [79] including the $\mathcal{O}(\alpha^2)$ and $\mathcal{O}(\alpha^3)$ contributions as given in Appendix A of Ref. [80]. Following the nomenclature in Ref. [79], <pdf_set> can be set to LO_beta, LO_eta, LO_mixed, and LO_collinear. The value of <pdf_scheme> must be chosen accordingly and set to beta, eta, mixed, and collinear, respectively. The starting scale μ_0 on which the LO_collinear case depends can be set via the input tag <pdf_starting_scale>, which has the default value 0.511×10^{-3}.

Warning

We note that lepton PDFs have not been thoroughly tested.

The parameters <helicity_em> and <helicity_ep> define the helicities of the incoming electrons and positrons, respectively. They can take the values ±1 for polarised beams. A value 0 corresponds to an unpolarised beam.

<dipoles> This section is only relevant for the real and integrated-dipole contributions. Here, the so-called α-parameters that control the phase-space volume of the subtraction and integrated dipoles can be set (see Ref. [81]). The α-parameters can be independently chosen using different input tags for some classes of emitter–emissus dipoles, specifically for final-state–final-state (`<alpha_fs_fs>`), final-state–initial-state (`<alpha_fs_is>`), initial-state–final-state (`<alpha_is_fs>`), and initial-state–initial-state (`<alpha_is_is>`) dipoles. The α-parameters can take values in the interval $(0, 1]$ and their common default value is 1. Note that no α-parameters are available for subtraction dipoles related to internal resonances in the pole approximation.

Corresponding real and integrated-dipole contributions must be calculated using the same value for the α-parameter, and the sum of these contributions must be independent of the specific value. Thus, the correctness of the subtraction procedure can be tested by varying the α-parameters.

A further parameter controls the subtraction of collinear singularities. By choosing `<DIS_scheme>` $= 1$, the DIS factorisation scheme is switched on for photon PDFs, while the default is to use the $\overline{\text{MS}}$ factorisation scheme throughout. The flag `<DIS_scheme>` has no effect for strongly interacting particles.

<integration_termination> This section allows to set conditions for a regular termination of the integration. The integration terminates upon fulfilling at least one of the following conditions. The integration stops once the number of accumulated accepted, generated, or vanishing events exceeds the values given via the parameters `<n_target_accepted_events>`, `<max_generated_events>`, and `<max_vanishing_events>`. Here, an event is vanishing when its weight is 0, while we refer to accepted events as those that pass the cuts (see Sec. 7.3). If the input value for `<n_target_accepted_events>` is lower than 1000, it is replaced by 1000 and a corresponding message appears on the standard output. To ease legibility for the values of these three parameters, spaces can be used to separate groups of digits.

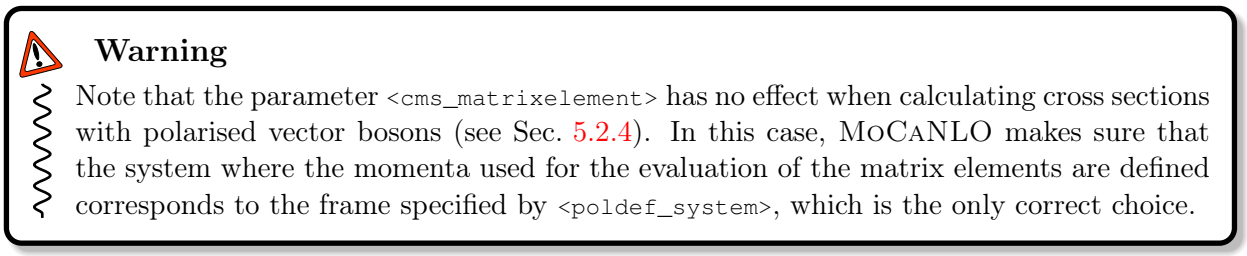
A run is also terminated after a certain relative or absolute precision has been reached. These are specified with the tags `<target_relative_precision>` and `<target_absolute_precision>`, whose values represent negative powers of ten. In Listing 16, for example, the target relative and absolute precisions are 10^{-5}. Finally, the termination of the integration is triggered by exceeding the run time specified by the `<time_wall>` parameter, whose value must have the format `days-hours:minutes`. This option can be useful when running on computer clusters with CPU-time limits. If no value is given to `<time_wall>`, this termination condition is not employed.

Table 11 summarises the parameters that control the MC integration and should only affect the results within integration errors.

The matrix elements can be calculated either in the laboratory system or with momenta boosted to the CM frame to improve numerical accuracy. This behaviour is controlled by the parameter `<cms_matrixelement>`. By default, `<cms_matrixelement>` $= 1$ and the CM-frame momenta are employed. By setting the `<cms_matrixelement>` $= 0$, the use of laboratory frame momenta can be enforced.

parameter	possible values	default value
<cms_matrixelement>	0, 1	1
<output>		(group)
<output_level>	2, 4, 6, 8	2
<print_feynman_diagrams>	true, false	true
<intermediate_output>		(group)
<intermediate_result_start>	integer	1000
<intermediate_result_stride>	integer	1000000
<intermediate_result_factor>	positive real number	1.1
<phase_space_generator>		(group)
<use_weight_optimization>	true, false	true
<channel_permutation>	0, 1	0
<channel_elimination>	0, 1	0
<subtraction_channels>	0, 1	0
<me_virt_veto_threshold>	positive real number	none
<pdf_mapping_emin>	positive real number	0.0
<minimum_relative_invariant>	positive real number	10^{-10}
<cos_theta_tolerance>	positive real number	10^{-10}
<tmax_tolerance>	positive real number	6.4×10^{-3}
<momentum_conservation_tolerance>	positive real number	10^{-12}
<onshellness_tolerance>	positive real number	10^{-12}
<running_options>		(group)
<store_run_resuming_info>	0, 1	0
<writing_mode>	0, 1	0
<load_run_resuming_info>	0, 1	0
<folder_overwriting>	0, 1, 2	0

Table 11: Elements of the <run_parameters> section of param_card.xml affecting the details of the MC integration and their defaults.



<output> The parameter <output_level> controls the amount of information for the standard output. The values <output_level> = 2, 4, 6, 8 correspond to minimal, intermediate, maximal relevant, and complete output, respectively.

By default, MoCaNLO prints two files collecting all Feynman diagrams that were used to construct the integration channels. The file `feynman_diagrams.tex` in the subdirectory `run.../data/init` contains the channels sorted in the order they were generated, while the file `feynman_diagrams_survey.tex` in the subdirectory `run.../data/run` contains them sorted by their importance in the integration. For complicated processes, these files can become very large and their writing very time consuming. The writing of these files can be switched off by issuing `<print_feynman_diagrams>` = `false`.

`<intermediate_output>` The parameters of this section control when intermediate results of the integration are written out. The parameter `<intermediate_result_start>` specifies after how many accepted events this occurs for the first time. Its default value is 1000. After the first time, intermediate results are written whenever the number of accepted events has increased by the value of `<intermediate_result_stride>`, with default 1000000, or by a factor corresponding to the value of `<intermediate_result_factor>`, with default 1.1.

`<phase_space_generator>` This section includes parameters that are relevant for the phase-space generation.

The parameter `<use_weight_optimization>` turns on or off the use of MoCaNLO's adaptive multi-channel weight-optimisation algorithm and should normally be set to `true`.

Several flags exist to control the selection of integration channels:

- If `<channel_permutation>` = 1, extra integration channels with permuted resonances are included. This is needed for processes with nested resonances, for example.
- If `<channel_elimination>` = 1, integration channels that differ from others only by internal non-resonant lines are eliminated. This can be useful to reduce the number of channels, in particular if `<channel_permutation>` = 1.
- The flag `<subtraction_channels>` is relevant only for real contributions. If its value is 1, extra integration channels corresponding to Catani–Seymour dipole subtraction contributions are taken into account. This can significantly increase the total number of channels. Note that no extra channels are available for subtraction dipoles related to internal resonances in the pole approximation.

The real parameter `<me_virt_veto_threshold>` can be set to discard events with unusually large virtual matrix elements, i.e. $2\,\mathrm{Re}\,\mathcal{M}_{\mathrm{virt}}\mathcal{M}^*_{\mathrm{Born}} >$ `<me_virt_veto_threshold>` $\cdot\,|\mathcal{M}_{\mathrm{Born}}|^2$. Such events might result from phase-space points that are problematic for Collier, but whose removal does not impact the final result, as long as their number is limited. It is the user's responsibility to ensure that only a few events are discarded when this parameter is activated, for instance by inspecting the output file `cross_section.dat` in the `result/` folder of the run.

A minimal partonic CM energy can be required to avoid the generation of events which would be cut away anyhow. This value can be specified via the parameter `<pdf_mapping_emin>`. We note that MoCaNLO also computes a minimal partonic CM energy from the transverse-momentum cuts provided via the `cut_card.xml` file (see Sec. 7.3). The value that is actually employed is the larger of the two. It needs to be larger than zero, otherwise the code stops with an error message.

The parameters `<minimum_relative_invariant>`, `<cos_theta_tolerance>`, `<tmax_tolerance>`, `<momentum_conservation_tolerance>`, and `<onshellness_tolerance>` are technical cut pa-

rameters of the MC integration that avoid the appearance of very large unphysical weights. They can be varied (typically by factors of 10) about their default values to verify that the integration results are independent of these parameters.

<running_options> This subsection contains a set of parameters that control the storing and loading of information during a computation, which can be used to resume a run after a walltime has been reached, for example. A description of the parameters in this subsection is given in Sec. 6.2.1.

6.2.1. How to perform sequential runs

MoCaNLO is capable of loading the results of a previous run and restarting a calculation that has met a termination criterion, provided the necessary information has been stored. In combination with the `time_wall` parameter (part of the `<integration_termination>` subsection of `param_card.xml`), this feature can be used to perform a resumption chain in which each step is within the walltime of a cluster, but which effectively corresponds to a longer integration runtime. By using this method, a smaller integration error can be achieved with the same CPU time compared to parallel job execution. This is because the optimised channel weights are inherited by subsequent steps of the resumption chain, whereas parallel runs perform the same channel weight optimisation independently. We remark that parallel and sequential running can be combined by simply computing parallel resumption chains.

The `<running_options>` subsection of the `<run_parameters>` section in the file `param_card.xml` contains a set of parameters that control the storing and loading of information during a computation. Listing 17 displays the complete set of parameters of this subsection.

```
<running_options>
   <store_run_resuming_info>  1 </store_run_resuming_info>
   <writing_mode>             0 </writing_mode>
   <load_run_resuming_info>   1 </load_run_resuming_info>
   <folder_overwriting>       1 </folder_overwriting>
</running_options>
```

Listing 17: MC running options that can be specified in the `run_parameters` section in `param_card.xml`.

Their possible and default values are found in the lower part of Table 11.

Setting `<store_run_resuming_info>` $= 1$ enables the storage of the information necessary to resume that run later. If `<store_run_resuming_info>` $= 0$, no storage will take place.

The parameter `<writing_mode>` controls at which points of the run the storage of information necessary for resumption takes place. By default (`<writing_mode>` $= 0$), the storage occurs when a MoCaNLO termination condition is met (see subsection `<integration_termination>` in Sec. 6.2). If `<writing_mode>` $= 1$, the storage occurs each time MoCaNLO writes intermediate results (see the beginning of Sec. 3.3).

Setting `<load_run_resuming_info>` $= 1$ tells the program that the user intends to resume a run and that it should load the necessary information. In this case, the run directory of the run that is to be resumed must be passed as an argument (see the example below for further details). If `<load_run_resuming_info>` $= 0$ (default), no loading will take place. By choosing both `<load_run_resuming_info>` $= 1$ and `<store_run_resuming_info>` $= 1$, intermediate steps in a chain of resumptions can be performed.

The parameter <folder_overwriting> determines how the results of a resumption run are stored, which is relevant for subsequent run resumptions. The default value <folder_overwriting> = 0 makes a backup of the results of the run that will be resumed and stores the results of the resumption under the name of the original run directory. If one sets <folder_overwriting> = 1, the results of previous steps of a resumption chain are kept, but the results of a step are overwritten if that step is performed again. Finally, the value <folder_overwriting> = 2 prevents any overwriting, so that the results of all intermediate and parallel steps are kept. An exemplary usage of this parameter can be found at the end of this subsection.

In the following, details on the storage and resumption procedures are given using the example introduced in Sec. 3.3. There, a MoCaNLO run for a specific subprocess was started

```
mocanlo  /path/to/process/pp_epvemumvmxbbx_qcd  1
```

where the two arguments are the path to the process directory, which contains the cards directory, and the run ID of the subprocess. If this run is performed with <store_run_resuming_info> = 0, the run's output directory contains the data subdirectory, with information about the initialisation and the run phase, as well as the result subdirectory, where cross sections and distributions are saved. If the information storage is activated by setting <store_run_resuming_info> = 1, an additional subdirectory named carbon_copy is created:

```
run_2025-07-29T17h38m28s928/
├── data/
│   └── ...
├── result/
│   └── ...
├── run_info.dat
└── carbon_copy/
    ├── channels_info.dat
    ├── cuts_info.dat
    ├── rnd_number_info.dat
    ├── run_id_info.dat
    ├── scale_factor_1/
    │   ├── cross_section_info.dat
    │   └── histograms/
    ├── scale_factor_2/
    │   ├── cross_section_info.dat
    │   └── histograms/
    └── ...
```

The files in carbon_copy contain the required information for the run resumption. In the file run_id_info.dat general information on how the run was performed is written, like its run_type, run_parameters, and any other tag specified for the run ID in the run_card.xml. This information is used when resuming a run to check that the code is restarted with the same set of parameters.

In rnd_number_info.dat, a set of 25 random numbers is stored that uniquely specifies the current point in the RANLUX pseudo-random number sequence [82]. Using this information, the event generation chain can restart from the exit point of the previous run. In channels_info.dat, information on the integration channels is stored, like the weight-optimisation step, together with the

channels' weights and variances, the respective contribution to the integrals and so on. This way, the program can benefit from a run in which an optimisation of the weights has occurred.

In order to consistently restore a run, information on the hits of the different cuts specified in `cut_card.xml` is saved in `cuts_info.dat`. This allows the cumulative counting of the hits for each cut.

Depending on the number of scales specified by `<scale_factors_f>` and `<scale_factors_r>`, an equal number of scale subdirectories is stored within the `carbon_copy` subdirectory (if only one scale is provided, no additional scale directory is produced). Therein, one can find all information to restart the cross section computation at the integrated level, in `cross_section_info.dat`, and even differentially, in the subdirectory `histograms`, containing information for all required distributions in separate files named after the observable name specified by the user. Moreover, in `cross_section_info.dat`, the maximum weight and the sum of the absolute weights computed so far are also contained.

In order to resume a previous run that contains a `carbon_copy` subdirectory, like the one of our example, whose output has been stored in `run_2025-07-29T17h38m28s928`, one has to set the parameter `<load_resuming_run_info>` to `1` and start a new run as

```
mocanlo  /path/to/process/pp_epvemumvmxbbx_qcd  1  \
         /path/to/run_2025-07-29T17h38m28s928
```

where the second argument is the run ID of the subprocess in question and the third argument is the path to the directory that contains the results of the run to be resumed.

An additional argument containing the random number seed before or after the name of the directory to resume is tolerated but not used, since all initialisation conditions for the random number chain and the integral computation are loaded from `run_2025-07-29T17h38m28s928/carbon_copy`. If a run is restarted with a different set of relevant parameters, the code stops with an error message pointing to a file containing the conflicting lines of the cards that triggered the error. This file is located in the subdirectory `data/init/user_input/` of the current run and named `diff_user_input_`*section*`.dat`, where *section* can be `cuts`, `histograms`, `monte_carlo`, `on_shell_projection`, `partonic_process`, `recombinations`, `resonances`, `scales`, `standard_model`, or `weight_opts` depending on which section caused the error. These files are temporary and are removed if the run resumption succeeds. The parameters contained in the subsections `<running_options>` and `<integration_termination>` of `param_card.xml` are ignored when comparing the input parameters of subsequent runs, so that their values can change without triggering an error.

The options `<load_run_resuming_info>` and `<store_run_resuming_info>` can be activated at the same time to perform chains of run resumptions. Depending on the value of `<folder_overwriting>` (discussed above), the name of the directory containing the results of a run resumption might not contain the time at which the directory was created, as opposed to the case of a fresh run. Therefore, this information is printed into a `run_info.dat` file, together with the name of the directory from which the data for the resumption was loaded. This file can be found directly in the directory of the resuming run, at the same level of the `data`, `results` and `carbon_copy` subdirectories.

The parameter `<folder_overwriting>` controls the location of the output of a resumed run. Setting `<folder_overwriting>` $= 2$ and executing the restart command above, the results of the restarted run are located in `run_2025-07-29T17h38m28s928_step1_2025-07-30T17h42m25s330`. Note that the name of this run directory comprises the original directory's name from which the data

have been loaded, the number of times N the run has been resumed (as stepN), and a new time stamp with the time at which the run has been restarted. By executing again the restart command above, a directory with a different time stamp,
e.g. run_2025-07-29T17h38m28s928_step1_2025-07-30T18h14m01s811, is created, containing results from a new resumption of the same run, thus carrying a different (second) timestamp.

If the flag <store_run_resuming_info> was set to 1, this run can be further resumed by running

```
mocanlo  /path/to/process/pp_epvemumvmxbbx_qcd  1  \
         /path/to/run_2025-07-29T17h38m28s928_step1_2025-07-30T18h14m01s811
```

which creates run_2025-07-29T17h38m28s928_step2_2025-07-31T18h17m28s388 directory, for instance. Note that the timestamp of the original run (step 0) is kept in the name at the first position, while the second timestamp corresponds to the last performed step (step2 in this case). The directory run_2025-07-29T17h38m28s928_step1_2025-07-30T18h14m01s811, which contains the results of the intermediate step1, as well as the directory run_2025-07-29T17h38m28s928_step1_2025-07-30T17h42m25s330, which contains the results of a step1 that were not used for a run resumption, remain.

In order to avoid the proliferation of intermediate-result directories, <folder_overwriting> can be set to 1. This causes the result directory of stepN to be replaced if a new stepN, i.e. the same step, is performed using the same loaded information. This option also affects the name of the result directories, in that no second timestamp is included. In our example, setting <folder_overwriting> = 1 and executing the first restart command stores the results into a folder run_2025-07-29T17h38m28s928_step1. Re-executing the command again will overwrite the step1 directory. Resuming the step1 of the run with the command

```
mocanlo  /path/to/process/pp_epvemumvmxbbx_qcd  1  \
         /path/to/run_2025-07-29T17h38m28s928_step1
```

creates a directory run_2025-07-29T17h38m28s928_step2 and so on. The directory run_2025-07-29T17h38m28s928_step1 remains. Therefore, only the most recent results of each step are stored.

By setting <folder_overwriting> = 0, the storage can be restricted to the two directories containing the results of the last two steps. By executing the first restart command, the original directory run_2025-07-29T17h38m28s928 is renamed into run_2025-07-29T17h38m28s928_bckp and the results of this first resumption are saved under the previous name of the original directory, i.e. run_2025-07-29T17h38m28s928. Issuing the restart command again, the second step is performed and its results are saved under the name run_2025-07-29T17h38m28s928, while run_2025-07-29T17h38m28s928_bckp now contains the results of the first step.

6.2.2. Photon–photon ultra-peripheral collisions

MoCaNLO is able to simulate initial-state photons originating from heavy ions in ultra-peripheral collisions. To that end, it relies on the external program gamma-UPC [71].

To steer all the possible options of the library, several flags are available in param_card.xml in the <pdfs> section. An example to run Pb–Pb collisions using the electric-dipole form factor (EDFF) [83] (instead of the charge (ChFF) form factor [84]) is:

parameter	possible values	default value
<alpha_UPC>	positive real number	1/137.036
<UPC_photon_flux>	0, 1	0
<nuclear_A_beam1>	integer	208
<nuclear_A_beam2>	integer	208
<nuclear_Z_beam1>	integer	82
<nuclear_Z_beam2>	integer	82

Table 12: Additional parameters for ultra-peripheral collisions in the param_card.xml and their default values.

```
<pdfs>
   <beam1>              ion          </beam1>
   <beam2>              ion          </beam2>
   <alpha_UPC> 7.2973525205055605d-3 </alpha_UPC>       ! alpha used in UPC
   <UPC_photon_flux>    0            </UPC_photon_flux> ! (0) EDFF scheme, (1) ChFF scheme
   <nuclear_A_beam1>    208          </nuclear_A_beam1> ! Case of Pb
   <nuclear_A_beam2>    208          </nuclear_A_beam2> ! Case of Pb
   <nuclear_Z_beam1>    82           </nuclear_Z_beam1> ! Case of Pb
   <nuclear_Z_beam2>    82           </nuclear_Z_beam2> ! Case of Pb
</pdfs>
```

The value of α used in UPC can be changed independently of the one used in the matrix element. The default values of the UPC parameters can be found in Table 12.

6.3. Setting a-priori weights for the Monte Carlo integration

By default, the a-priori weights of the different MC integration channels are all set to the same value, equal to the inverse of the total number of integration channels for the process of interest. They represent initialisation values for the multi-channel optimisation, which adapts them during the integration [85]. In order to improve the convergence of the result for processes dominated by diagrams containing resonances, it can be useful to tune the a-priori weights at initialisation. This can be achieved by linking in the run_card.xml a section <weight_opts id="XX">, where XX is an ID that denotes a particular a-priori weight setting. This section contains one or more subsections <weight_opt> that independently specify the information on the resonant contributions to be enhanced. Therefore, the a-priori weights of the integration channels corresponding to Feynman diagrams containing at least <min_n_resonant_particles> and at most <max_n_resonant_particles> particles specified by <particle> as an s-channel resonance are adjusted. They are multiplied by a factor <mult_factor> for each resonance found in the associated Feynman diagram. The global tag <total_max_n_resonant_particles> limits the total number of enhancement factors to the specified number. Thus, a channel with more resonances than <total_max_n_resonant_particles> is not enhanced at all.

An exemplary application is shown in Listing 18 for tZj production, where a-priori weights can be enhanced by up to a factor of 30, i.e. by a factor of 10 if they belong to integration channels corresponding to diagrams with exactly one top quark, and by an additional factor of three if their integration channels also include one Z boson. Note that in the example there are no a-priori weights that can be multiplied by a factor higher than 30, as the global tag <total_max_n_resonant_particles> limits the number of enhancement factors to 2. Therefore, there are four distinct situations in which

a-priori weights can be enhanced: for channels with one top quark (multiplied by 10), one Z boson (multiplied by 3), two Z bosons (multiplied by 9), and one top quark and one Z boson (multiplied by 30).

```
<weight_opts id="tzj">
   <total_max_n_resonant_particles> 2   </total_max_n_resonant_particles>
   <weight_opt>
      <particle>                    t   </particle>
      <min_n_resonant_particles>    1   </min_n_resonant_particles>
      <max_n_resonant_particles>    1   </max_n_resonant_particles>
      <mult_factor>                 10  </mult_factor>
   </weight_opt>
   <weight_opt>
      <particle>                    z   </particle>
      <min_n_resonant_particles>    1   </min_n_resonant_particles>
      <max_n_resonant_particles>    2   </max_n_resonant_particles>
      <mult_factor>                 3   </mult_factor>
   </weight_opt>
</weight_opts>
```

Listing 18: A-priori weights in `param_card.xml`.

Few additional comments on the `<particle>` tag are in order. First of all, note that particles and antiparticles are distinguished. In order to treat them on the same footing, the MoCaNLO identifier of the resonance (see second column of Table 4) must be accompanied by a star key, i.e. `t*` indicates either a `t` or a `t~`, and similarly `w*` either a `w+` or a `w-`. Moreover, a list of particles can be specified in `<particle>`: particles in the list are treated on the same level as far as the assignment of an enhancement factor of the a-priori weight is concerned.

7. Recombination, cut, and scale definitions: the `cut_card.xml`

The `cut_card.xml` defines the sections `<recombinations>`, `<cuts>`, and `<scales>` for setting the recombination parameters for the clustering algorithm, defining the acceptance function, and fixing the dynamical scales, respectively.

After recombination, clustered objects are stored in *sorted clusters*, which consist of lists of different jet types, each ordered by transverse momentum (see Sec. 7.3). The cut procedure operates on these sorted clusters by modifying and extending them. The definition of scales and of observables of histograms is based on the sorted clusters that result from the cut procedure.

7.1. Recombinations

The settings in the section `<recombinations>` of the `cut_card.xml` configure the jet algorithm. The scheme allows for different steps of recombinations, which can be executed consecutively or in parallel. The recombination may involve several levels, at which the parallel recombinations are merged. The input of the recombination is the final-state particles of the considered process, while the output is a list of clusters, to be used as input for the cut routines. Note that the final list of clusters includes only objects that pass through all recombination levels, even if they are not subject to recombination.

The parameters, with their possible and default values, of the section `<recombinations>`, the subsections `<recombination_step>`, and their subsections `<recombination>` are listed in Tables 13, 14, and 15, respectively. A simple example with only one recombination step and thus one recombination

Parameter	possible values	default value
<rapidity_maximum>	real number	100.0
<recombination_index>	real number	-1
<recombination_step>	—	—

Table 13: Parameters and possible values of the <recombinations> section.

Parameter	possible values	default value
<name>	string	none
<recombination_index>	real number	-1
<recombination_level_in>	non-negative integer	0
<recombination_level_out>	positive integer	1
<recombination_jet_types_in>	list of jet types	none
<recombination_jet_types_out>	list of jet types	none
<recombination>	—	—

Table 14: Parameters and possible values of the <recombinations_step> section.

level is shown in Listing 19,

```
<recombinations id="ttx">
    <rapidity_maximum>              5   </rapidity_maximum>
    <recombination_step>
        <recombination_index>      -1   </recombination_index>
        <recombination_level_in>    0   </recombination_level_in>
        <recombination_level_out>   1   </recombination_level_out>
        <recombination type="different_type">
            <name> bjet-jet </name>
            <recombinant>           2   </recombinant>
            <recombinendus>         1   </recombinendus>
            <minimum_distance>      0.5 </minimum_distance>
        </recombination>
        <recombination type="same_type">
            <name> bjet-bjet </name>
            <recombinendus1>        2   </recombinendus1>
            <recombinant>           1   </recombinant>
            <minimum_distance>      0.5 </minimum_distance>
        </recombination>
    </recombination_step>
</recombinations>
```

Listing 19: Example recombinations in cut_card.xml.

while a more complicated example with three recombination steps and two recombination levels is given in Listing 20.

```
<recombinations id="vbs_semileptonic">
    <rapidity_maximum>              5   </rapidity_maximum>
    <recombination_step>
        <name> lepton-photon </name>
        <recombination_index>       0   </recombination_index>
        <recombination_level_in>    0   </recombination_level_in>
        <recombination_level_out>   1   </recombination_level_out>
        <recombination type="different_type">
            <name> antimuon-photon </name>
```

Parameter	possible values	default value
<name>	string	none
<recombinant>	integer	0
<recombinendus>	integer	0
<recombinendus1>	integer	<recombinant>
<minimum_distance>	real number	0.0
<jet_type_selector>	real number	0.0
<jet_isolation_distance>	real number	0.0
<jet_isolation_parameter>	real number	0.0
<jet_isolation_exponent>	real number	1.0

Table 15: Parameters and possible values of the <recombination> section.

```
            <recombinant>           16  </recombinant>
            <recombinendus>         3   </recombinendus>
            <minimum_distance>      0.1 </minimum_distance>
        </recombination>
    </recombination_step>
    <recombination_step>
        <name> slim-jets </name>
        <recombination_index>      -1   </recombination_index>
        <recombination_level_in>    1   </recombination_level_in>
        <recombination_level_out>   2   </recombination_level_out>
        <recombination type="same_type">
            <name> jet-jet </name>
            <recombinant>           1   </recombinant>
            <minimum_distance>      0.4 </minimum_distance>
        </recombination>
        <recombination type="different_type">
            <name> jet-photon </name>
            <recombinant>           1   </recombinant>
            <recombinendus>         3   </recombinendus>
            <minimum_distance>      0.4 </minimum_distance>
        </recombination>
    </recombination_step>
    <recombination_step>
        <name> fat-jets </name>
        <recombination_index>      -1   </recombination_index>
        <recombination_level_in>    1   </recombination_level_in>
        <recombination_level_out>   2   </recombination_level_out>
        <recombination_jet_types_in>    1 3 </recombination_jet_types_in>
        <recombination_jet_types_out> 31 3 </recombination_jet_types_out>
        <recombination type="same_type">
            <name> jet8-jet8-jet8 </name>
            <recombinant>           31  </recombinant>
            <minimum_distance>      0.8 </minimum_distance>
        </recombination>
        <recombination type="different_type">
            <name> jet-photon </name>
            <recombinant>           31  </recombinant>
            <recombinendus>         3   </recombinendus>
            <minimum_distance>      0.8 </minimum_distance>
        </recombination>
    </recombination_step>
</recombinations>
```

Listing 20: Example recombinations in cut_card.xml with multiple steps.

The section <recombinations id="XX">, linked via the ID in the run_card.xml, contains one or more <recombination_step> subsections as well as the further parameters <rapidity_maximum> and <recombination_index>. Particles with a rapidity larger than <rapidity_maximum> are disregarded in the recombination algorithm. Since neutrinos do not undergo any recombination, they are not affected by it. The parameter <recombination_index> defines the default clustering algorithm and takes the values -1, 0, and 1 for the anti-k_t [86], the Cambridge–Aachen [87, 88], and the k_t algorithm [89, 90], respectively.

The parameters of a <recombination_step> are summarised in Table 14. The optional tag <name> is only used for output purposes. The optional parameter <recombination_index> allows to overwrite, for the corresponding <recombination_step>, the value of <recombination_index> defined for the <recombinations id="XX"> section, discussed above. Note that only one algorithm can be chosen within a single <recombination_step>.

Since multiple steps of recombination are allowed, one must specify the level in the recombination chain on which a recombination step acts via the tag <recombination_level_in>. Recombination steps with the same <recombination_level_in> values are run in parallel. The tags <recombination_level_in> and <recombination_level_out> define the input and output level of the recombination step, respectively, and by default equal 0 (all particles) and 1. Internally, the clusters of level <recombination_level_in> are copied to clusters of level <recombination_level_out> (to allow for parallel recombination with different jet resolution parameters for instance), and thereafter the recombination step is performed on clusters of level <recombination_level_out>. Once all steps for an output level have been done, the resulting clusters are merged to a new list of clusters, which can serve as input for subsequent, not necessarily consecutive, levels. The merged clusters of the final level are the output of the recombination procedure.

The list <recombination_jet_types_in> allows to restrict the jet types on which the recombination step acts. If this list is specified, all recombined objects get the jet type given by the corresponding list <recombination_jet_types_out>. If <recombination_jet_types_in> is not specified or empty, all visible jet types, i.e. those within <rapidity_maximum>, and all neutrinos are used as input and, if not recombined, passed to the output. Note that each jet type that should appear in the final output of the recombination routine must also be present in the input at every recombination level. This requirement can be satisfied by defining a sequence of recombination steps without the <recombination_jet_types_in> parameter, thereby propagating all jet types through all levels.

Each recombination step necessarily contains different recombination rules, defined either for jets of the same type by the tag <recombination type="same_type"> or for jets of different type by the tag <recombination type="different_type">. An arbitrary name can be set for both types via <name>, which is only used for output purposes. The jet-resolution parameter is fixed by <minimum_distance>, corresponding to the rapidity–azimuthal-angle distance, and the jet type of the result of the clustering by <recombinant>.

For same-type-jet recombination, the type of the jets to cluster is defined by <recombinendus1>. If <recombinendus1> is not specified, it is assumed to be equal to <recombinant>. In the example in Listing 19, the same-type recombination with the name bjet-bjet defines the clustering of b jets (jet type 2) with a jet resolution parameter $R = 0.5$ into flavourless jets (jet type 1).

A different-type recombination requires the definition of two different types of jets to be recombined via <recombinendus1> and <recombinendus>. While <recombinendus> needs to be specified, <recombinendus1> is assumed to be equal to <recombinant>, if not given. In Listing 19, the different-

type recombination with the name `bjet-jet` defines the clustering of b jets and jets (jet type 1) into b jets with a jet resolution parameter $R = 0.5$.

Valid values for `<jet_type>` created in the recombination

The jet types appearing within recombination steps must correspond to existing jet types of MoCaNLO as given in Table 4 or new ones defined via `<recombination_jet_types_out>` or `<recombinant>` within the range 30–50.

In the example Listing 20, in the first recombination step photons are clustered with antimuons, using the Cambridge–Aachen algorithm with jet resolution parameter $R = 0.1$. This step passes all jets to level 1. At this level recombination steps 2 and 3 cluster jets of type 1 (and photons) twice in parallel using the anti-k_T algorithm with resolution parameters 0.4 (slim-jets) and 0.8 (fat-jets). While the slim jets keep their jet types (if not clustered), the fat jets get jet type 31 via the parameter `<recombination_jet_types_out>` = 31 before clustering. Note that also photons (jet type 3) have to be present in the lists `<recombination_jet_types_in>` and `<recombination_jet_types_out>` to allow their clustering with fat jets. At level 2, which is the last one in the example, the outputs of recombination steps 2 and 3 are internally merged. Thus, the lists of jets in the output of the recombination involve the same objects twice. This has to be rectified by using appropriate cuts.

7.2. Treatment of photons

The treatment of photons in the final state (via isolation, fragmentation, and conversion to jets) is determined to a large extent by the recombination rules.

7.2.1. Frixione isolation

Photons can be isolated via the Frixione isolation scheme [19]. To this end, a recombination rule as shown in Listing 21 should be added to the corresponding recombination step in the run card.

```
<recombination type="different_type">
    <name>                  jet-photon </name>
    <recombinant>               3      </recombinant>
    <recombinendus>             1      </recombinendus>
    <minimum_distance>          0.5d0  </minimum_distance>
    <jet_isolation_parameter>  0.11d0  </jet_isolation_parameter>
    <jet_isolation_exponent>    1d0    </jet_isolation_exponent>
    <jet_isolation_distance>    0.5d0  </jet_isolation_distance>
</recombination>
```

Listing 21: Frixione isolation in `cut_card.xml`.

If the rapidity–azimuthal-angle distance ΔR between the photon (=`<recombinant>`) and the jet (=`<recombinendus>`) is smaller than the cone size R_0 (=`<jet-isolation-distance>`), the two objects `<recombinant>` and `<recombinendus>` are discarded, unless they respect the relation

$$p_{\mathrm{T,recombinendus}} < \epsilon p_{\mathrm{T},\gamma} \left(\frac{1 - \cos(\Delta R_{\gamma,\mathrm{recombinendus}})}{1 - \cos(R_0)} \right)^n , \tag{3}$$

where $p_{\mathrm{T,recombinendus}}$ and $p_{\mathrm{T},\gamma}$ are the transverse momenta of the jet and the photon, respectively. If $\Delta R < R_0$ and Eq. (3) is fulfilled, the two objects are merged into one of type `<recombinant>`,

i.e. a photon. For $\Delta R > R_0$, both objects are kept independently of Eq. (3). The prefactor ϵ (=<jet_isolation_parameter>) controls the allowed range of the energy fraction of the recombinendus in the cone around the photon and the exponent n (=<jet_isolation_exponent>) the approach to the limit. Both should be positive parameters of order one and have default values <jet_isolation_parameter> = 0 and <jet_isolation_exponent> = 1.

Restriction

The use of Frixione isolation via <jet_isolation_parameter>, <jet_isolation_exponent>, and <jet_isolation_distance> is only implemented for the case of a photon as <recombinant> and a jet originating from a massless parton as <recombinendus>, more precisely for <jet_type> $\in \{1, 2, 30, \ldots, 50\}$, where jets of type 30–50 must be defined appropriately.

7.2.2. Quark-to-photon fragmentation function

Alternatively, isolated photons can be treated by employing a cut on the energy fraction of the photon within a recombined quark–photon pair. To this end, a recombination rule as shown in Listing 22 should be added to the corresponding recombination step in the run card.

```
<recombination type="different_type">
    <name>               jet-photon </name>
    <recombinant>        3          </recombinant>
    <recombinendus>      1          </recombinendus>
    <minimum_distance>   0.5d0      </minimum_distance>
    <jet_type_selector>  0.1d0      </jet_type_selector>
</recombination>
```

Listing 22: Use of fragmentation function for final states with photons in cut_card.xml.

If <jet_type_selector> = 0.0 (default), <recombinant> and <recombinendus> are recombined to <recombinant> as usual. For <jet_type_selector> > 0.0, the recombination is modified as follows. If the energy fraction of the <recombinendus>

$$z = \frac{E_{\text{recombinendus}}}{E_{\text{recombinendus}} + E_{\text{recombinant}}}$$

is larger than z_{cut} (=<jet_type_selector>), both jets are recombined into a jet of type <recombinendus>, otherwise they are recombined into a jet of type <recombinant>.

A recombination depending on the energy fraction of the <recombinendus> must be accompanied by a fragmentation-function contribution in order to ensure IR-finite results. A quark-to-photon fragmentation function has been introduced in Refs. [20, 91] and the non-perturbative fit parameters entering it were obtained in Ref. [92] and read

$$\mu_0 = 0.14\,\text{GeV} \qquad \text{and} \qquad C = -13.26. \tag{4}$$

The quark-to-photon fragmentation function has been implemented in MoCaNLO following Refs. [21, 26, 93–95].

The quark-to-photon fragmentation function is needed in two different situations. The first case appears in the calculation of EW corrections (real photon radiation) to a LO process containing

a final-state gluon. An example is the process pp → Zj [94], which contains the partonic process $q\bar{q} \to Zg$. When considering real photonic corrections, a hard photon can be recombined with a soft jet. Such a configuration leads to a QCD IR singularity that is not cancelled by any standard dipole but needs a quark-to-photon fragmentation function. The appropriate recombination rule for this case is given in Listing 23.

```
<recombination type="different_type">
    <name>               jet-photon </name>
    <recombinant>        1          </recombinant>
    <recombinendus>      3          </recombinendus>
    <minimum_distance>   0.1d0      </minimum_distance>
    <jet_type_selector>  0.7d0      </jet_type_selector>
</recombination>
```

Listing 23: Use of fragmentation function for final states with gluons in `cut_card.xml`.

If the energy fraction of the photon (`<recombinendus>`)

$$z_\gamma = \frac{E_{\text{recombinendus}}}{E_{\text{recombinendus}} + E_{\text{recombinant}}}$$

is larger than z_{cut} (=`<jet_type_selector>`), the two objects are recombined into an object of type photon, otherwise they are recombined into a jet of type `<recombinant>`. The real and integrated dipole-subtraction terms inherit the splitting cut z_{cut} from the recombination rule (parameter `<jet_type_selector>`) and the corresponding integrated dipoles, i.e. the ones with a fermionic emitter and a massless vector boson as emissus, take the quark-to-photon fragmentation function properly into account.

In the second case, QCD corrections are computed for a LO process containing a final-state photon. An example is the process pp → Zγ [96]. When considering QCD corrections, a soft photon can be recombined with a hard quark jet, for instance in the process ug → Zγu. This gives rise to a collinear singularity that is compensated by a contribution of the quark-to-photon fragmentation function in combination with the LO process ug → Zu, as discussed in Refs. [22, 96]. The appropriate recombination rule for this case is given in Listing 22. If the energy fraction of the jet (`<recombinendus>`)

$$z_{\text{j}} = \frac{E_{\text{recombinendus}}}{E_{\text{recombinendus}} + E_{\text{recombinant}}}$$

is larger than z_{cut} (=`<jet_type_selector>`), the objects are recombined into a jet of the type `<recombinendus>`, otherwise they are recombined into an object of type photon (`<recombinant>`). As in the previous case, the real and integrated dipole-subtraction terms inherit the splitting cut from the recombination rule (parameter `<jet_type_selector>`) and the integrated dipoles, i.e. the ones with a fermionic emitter and a massless vector boson as emissus, take the quark-to-photon fragmentation function properly into account.

Restriction

The quark-to-photon fragmentation function is only implemented for the case of a photon as `<recombinant>` and a massless quark, specifically with `PDG code` ≤ 5, as `<recombinendus>`.

7.2.3. Photon-to-jet conversion function

The second type of IR singularities associated with photons concerns the splitting of a virtual photon into a quark–antiquark pair, $\gamma^* \to q\bar{q}$, in the final state. These singularities are absorbed in MoCaNLO by using a photon-to-jet conversion function as discussed in detail in Refs. [23, 51]. In MoCaNLO the photon-to-jet conversion function is switched on by default in integrated dipoles for the splitting of a photon into a quark–antiquark pair. The implementation is based on the 5-flavour scheme, and the finite contributions are distributed according to their squared charges to the five light quarks. The value for $\Delta\alpha^{(5)}_{\rm had}(M_Z^2)$ is taken from Ref. [97] and hard-coded:

$$\Delta\alpha^{(5)}_{\rm had}(M_Z^2) = 276.11 \times 10^{-4}.$$

.

7.2.4. DIS factorisation scheme for photon PDFs

For photon PDFs in hadron collisions, the DIS factorisation scheme can also be used. It can be switched on adding the entry

```
<DIS_scheme> 1 </DIS_scheme>
```

to the `<run_parameters>` section of the `param_card.xml`. For the default value `<DIS_scheme>` $= 0$, the $\overline{\rm MS}$ factorisation scheme is used, while `<DIS_scheme>` $= 1$ turns on the DIS factorisation scheme only for photon PDFs.

7.3. Cuts

In MoCaNLO, the cuts are not only used for event selection, but they also serve to tag particles, order them, and create reconstructed objects. These cuts take the form of a list of operations to be applied sequentially to the final-state clusters resulting from the recombination. It is important to note that the order in which the cuts are applied is crucial.

The cuts operate on *sorted clusters*, which contain for each jet type a list of *recombinants*, resulting from the recombination. Recombinants, jets, and clusters are used as synonyms in the following, even if these can be any kind of clusters of particles or single particles like leptons. The recombinants are objects of a specific `<jet_type>`, as defined in column 5 of Table 4, or of new jet types within the range 30–50 generated by the recombination or cut procedure. Each jet in a list of recombinants has two additional attributes, a `<jet_tag>` and a `<jet_order>`, which might be changed by applying a cut. By default, after recombination and before applying any cut, `<jet_tag>` and `<jet_order>` are equal to 0, and the lists of recombinants of each `<jet_type>` are ordered according to their transverse momenta. The cuts do not change the entries in the list of recombinants, but only modify their associated parameters `<jet_tag>` and `<jet_order>`. Besides the lists of recombinants, sorted clusters contain two further lists for each `<jet_type>`: the *ordered recombinants* and the *reconstructed recombinants*, which are both initially empty. The reconstructed recombinants contain the reconstructed objects created by the *reconstruction cuts*, and the lists of ordered recombinants are filled subsequently whenever a cut *orders* jets.

As mentioned above, the cuts allow to tag and order the jets, to apply cuts on jets with specific `<jet_tag>` and `<jet_order>`, and to create new types of jets upon combining existing jets. Tagging

a jet sets the corresponding <jet_tag> to one. Ordering a jet adds this jet to the list of ordered recombinants and sets its <jet_order> to a value equal to the position of the object in the list of ordered recombinants for the corresponding <jet_type>. The same jet is still present in the list of recombinants, but now with parameter <jet_order> different from zero. Reconstruction cuts create new jets upon combining or cloning existing ones and add them to the lists of reconstructed recombinants. Upon ordering them, they also enter the corresponding lists of ordered recombinants.

Besides eliminating events and jets, the cuts define the jets that enter the histograms and the definition of dynamical scales. To this end, the usage of the lists of ordered recombinants is recommended, because only these are uniquely defined. Thus, all jets intended for histogram filling or scale definition should be tagged and ordered. Once a given type of clusters has been ordered, no further cuts must be applied that could remove or reorder any of the ordered jets of this type. Otherwise, subsequent access to these jets, for example by additional cuts, scale definitions, or histogram calls, may lead to run-time errors.

Even if a mechanism has been implemented to warn the user in case of inconsistencies in the input, not all problematic cases might be catched. In order to further check the correct implementation of the cuts in the `cut_card.xml`, it might be useful to compile the code with the flag `DEBUG_CUTS` switched on in `include/debug_flags.h`, to run it with a small number of events, e.g. using the tag <max_generated_events>, and to check whether the cuts act as intended.

After this introduction, we turn to a more detailed description of the cut implementation. In a typical application, at a given stage of the cut sequence, tagged jets (<jet_tag> = 1), which are distinguished by their order, tagged jets without an order (<jet_order> = 0), and untagged jets (<jet_tag> = 0), which are unordered, are present simultaneously. In addition, there can be discarded jets with <jet_tag> = −1. Some examples of cuts are shown in Listing 24 and explained in detail later in this section.

All cuts are defined in the section <cuts> in `cut_card.xml`. Each cut is set via the XML tag <cut> and can be given an arbitrary name with the tag <name>, which is only used for output purposes. Cuts always come with a type, which is defined by the tag attribute `type`. There are four classes of cuts: single-particle, pair-particle, multi-particle, and reconstruction cuts. All these are explained in the following.

Each cut acts on a single jet or on a set of jets, characterised by their type, tag, and order status. If no matching jets are found, the cut either has no effect or results in the removal of the event, depending on its definition. The latter is to a large extent determined by three more tags: the <target>, the <action>, and the <order> of a cut.

The <target> tag specifies the object the cut acts on. It can take the values <target> = 0, if acting on the whole event, <target> = 1, if on the jet of <jet_type> or <jet_type_1>, or <target> = 2, if on the jet of <jet_type_2> in a pair-particle cut, and even larger values in multi-particle or reconstruction cuts. By default, MoCaNLO sets <target> = 0 for all cuts apart from the reconstruction cuts, where the default is <target> = 1. In a reconstruction cut, for <target> = 2 the tagging applies to both lists <jet_type_1> and <jet_type_2> (if present), while the cut acts always on the last object (most right) in <jet_type_1>.

The tag <action> defines the action of the cut, with a default value <action> = −1. For <action> = −1 and <target> = 0, the event is discarded, while for <action> = −1 but <target> > 0, the corresponding target of the cut is eliminated, if the value of the corresponding observable is not between <min_value> and <max_value>. For instance, if <target> = 1, the jet of <jet_type> or

```
<cuts id="vbs">
    <cut type="transverse_momentum">
        <name>          jet_pt_cut </name>
        <jet_type>      1          </jet_type>
        <jet_tag>       0          </jet_tag>
        <jet_order>     0          </jet_order>
        <target>        1          </target>
        <action>        1          </action>
        <min_value>     30.0       </min_value>
        <n_required>    2          </n_required>
    </cut>
    ...
    <cut type="distance">
        <name> ep_tag_jet_distance_cut </name>
        <jet_type_1>    1          </jet_type_1>
        <jet_tag_1>     1          </jet_tag_1>
        <jet_order_1>   0          </jet_order_1>
        <jet_type_2>    4          </jet_type_2>
        <jet_tag_2>     0          </jet_tag_2>
        <jet_order_2>   0          </jet_order_2>
        <target>        1          </target>
        <action>        0          </action>
        <min_value>     0.4        </min_value>
        <n_required>    2          </n_required>
    </cut>
    ...
    <cut type="rapidity">
        <name> jet_rapidity_cut  </name>
        <jet_type>      1          </jet_type>
        <jet_tag>       1          </jet_tag>
        <jet_order>     0          </jet_order>
        <target>        1          </target>
        <action>        0          </action>
        <order>         1          </order>
        <max_value>     4.7        </max_value>
        <n_required>    2          </n_required>
    </cut>
    ...
    <cut type="invariant_mass ">
        <name> invariant_mass_tag_jets </name>
        <jet_type>      1 1        </jet_type>
        <jet_tag>       1 1        </jet_tag>
        <jet_order>     1 2        </jet_order>
        <min_value>     400.0      </min_value>
    </cut>
</cuts>
```

Listing 24: Cuts in `cut_card.xml`.

<jet_type_1> is removed, i.e. its parameter <jet_tag> is set to -1, if the value of the corresponding observable is outside the cut range. For <action> $= 1$ and <target> $= 0$, the cut acts as a veto, i.e. if the variable is within the cut range the event is discarded. For <action> $= 1$ and <target>$= 1$, the jet is tagged, i.e. its parameter <jet_tag> is set to 1, if it passes the cut, i.e. if the value of the corresponding observable is in the range between <min_value> and <max_value>. Conversely, for <action> $= 0$ and <target> $= 1$, the jet is untagged, i.e. its parameter <jet_tag> is set to 0, if it does not pass the cut. The tagging and untagging acts on all jets with specified <jet_type>, <jet_tag>, and <jet_order>. For <action> $= -1$, all jets with specified <jet_type> and <jet_order>, but with tag parameter greater or equal to <jet_tag>, are discarded.

Finally, cuts also come with an <order> tag, whose meaning depends on the cut type. All single-particle cuts with the tag <order> $= 1$ and pair-particle cuts with the tag <order_1> $= 1$ or

Parameter	possible values	default value
<name>	string	none
<jet_type>	$1, 2, \ldots, 50$	0
<jet_tag>	$0, 1$	0
<jet_order>	$0, 1, \ldots$	0
<target>	$0, 1$	0
<action>	$-1, 0, 1$	-1
<order>	$0, 1$	0
<n_required>	$0, 1, \ldots$	0
<min_value>	real number	0.0
<max_value>	real number	huge(0.0)

Table 16: Parameters and possible values of single-particle cuts. For a cut of type `rapidity` or `pseudorapidity`, the default value of <min_value> is the negative of <max_value>.

<order_2> $= 1$ order all respective tagged jets after application of the cut by setting their parameter <jet_order> to consecutive values larger than zero. The default is <order> $= 0$, which means do not order. Multi-particle cuts and reconstruction cuts with <order> $= 1$ set the order of the tagged jets to the number of previously tagged jets plus one.

Ordering of jets

Ordering via single- or pair-particle cuts must be done only once per jet type, since these order all tagged jets of the respective type. Tagging via multi-particle cuts and reconstruction cuts should only be done when no re-ordering via single- or pair-particle cuts follows thereafter. For reconstruction cuts, which should normally be the last cuts to be applied, <order> $= 1$ implies ordering of all jets tagged by this cut, even if these were tagged before.

A **single-particle cut** is defined for the observable of a single particle, e.g. a cut on its transverse momentum. The corresponding parameters, possible values, and default values are summarised in Table 16. A single-particle cut requires the tag <jet_type> specifying the single jet the cut is applied to, a minimum and/or a maximum value in <min_value> and/or <max_value>, respectively, and the number of required particles <n_required> of this jet type in the final state. Optionally, <jet_tag> and <jet_order> can be specified. Otherwise, these are assumed to be zero, meaning that the cut applies to all untagged and unordered jets. For a sensible cut, at least the tags <jet_type> and either <min_value> or <max_value> have to be provided. The action of single-particle cuts is summarised in Table 17.

Available single-particle cut types are:

- `transverse_momentum`, $p_\mathrm{T} = \sqrt{p_x^2 + p_y^2}$,
- `energy`, $E = p_0$,
- `rapidity`, $y = \frac{1}{2} \ln \left(\frac{p_0 + p_z}{p_0 - p_z} \right)$,

<target>	<action>	**result**
0	−1	remove event if less than <n_required> jets pass the cut
0	1	remove event if the cut variable for at least one jet is within <min_value> and <max_value> (veto). (parameter <n_required> not allowed!)
1	1	tag jet if it passes the cut and remove event if less than <n_required> tagged jets pass the cut
1	0	untag jet if it does not pass the cut and remove event if less than <n_required> tagged jets pass the cut
1	−1	remove jet if it does not pass the cut and remove event if less than <n_required> (tagged or untagged) jets pass the cut. Jets are removed irrespective of their tagging status.

<order>	**result**
1	order tagged jets according to p_T and add them to the list of ordered recombinants

Table 17: Action of single-particle cuts.

- `pseudorapidity`, $\eta = \frac{1}{2}\ln\left(\frac{|\mathbf{p}|+p_z}{|\mathbf{p}|-p_z}\right)$,
- `theta_angle`, $\frac{180}{\pi}\arccos\left(\frac{p_z}{|\mathbf{p}|}\right)$,
- `single_invariant_mass`, $M=\sqrt{p^2}$.

Note that the cut `single_invariant_mass` acts on the invariant mass of a single jet.

In the example in Listing 24, the first cut acts on untagged and unordered jets of <jet_type> = 1 and requires a transverse-momentum of $p_{\mathrm{T,j}} > 30\,\mathrm{GeV}$. It tags jets that fulfil this criterion and requires two tagged jets in the final state. The third cut in Listing 24 is a rapidity cut on tagged jets with $|y_\mathrm{j}| < 4.7\,\mathrm{GeV}$, which untags jets and orders the remaining tagged jets.

A **pair-particle cut** is defined for an observable of two clusters, e.g. their ΔR_{ij} distance or their invariant mass m_{ij}. A pair-particle cut requires two jet types via <jet_type_1> and <jet_type_2>, possibly being the same type, corresponding jet tags <jet_tag_1> and <jet_tag_2>, and jet orders <jet_order_1> and <jet_order_2>. Jet tags and orders that are not specified are assumed to be zero. The associated cut observable is evaluated for all cluster pairs with the two jet types, jet tags, and jet orders in the final state, excluding pairs of identical clusters. The complete set of parameters for pair-particle cuts is listed in Table 18, while the actions of the cuts are summarised in Table 19. The parameter <n_required> is not allowed for <target> = 0. For <target> = 1 or <target> = 2, <n_required> applies to <jet_type_1> or <jet_type_2>, respectively. Tagging and ordering are restricted to jets within <jet_type_1> for <target> = 1 and to jets within <jet_type_2> for <target> = 2.

Available pair-particle cut types are:

- `distance`, $\Delta R_{12} = \sqrt{(y_1-y_2)^2+(\Delta\phi)^2}$, $\Delta\phi = \min(|\phi_1-\phi_2|, 2\pi-|\phi_1-\phi_2|)$,

Parameter	possible values	default value
<name>	string	none
<jet_type_1>	$1, 2, \ldots, 50$	0
<jet_tag_1>	$0, 1$	0
<jet_order_1>	$0, 1, \ldots$	0
<jet_type_2>	$1, 2, \ldots, 50$	0
<jet_tag_2>	$0, 1$	0
<jet_order_2>	$0, 1, \ldots$	0
<target>	$0, 1, 2$	0
<action>	$-1, 0, 1$	-1
<order_1>	$0, 1$	0
<order_2>	$0, 1$	0
<n_required>	$0, 1, \ldots$	0
<min_value>	real number	0.0
<max_value>	real number	huge(0.0)

Table 18: Parameters and possible values of pair-particle cuts.

- `pseudodistance`, $\Delta r_{12} = \sqrt{(\eta_1 - \eta_2)^2 + (\Delta\phi)^2}$,
- `pair_invariant_mass`, $M_{12} = \sqrt{(p_1 + p_2)^2}$,
- `pair_transverse_momentum`, $p_{\mathrm{T},12} = \sqrt{(p_{x,1} + p_{x,2})^2 + (p_{y,1} + p_{y,2})^2}$,
- `angular_separation`, $\Delta\theta_{12} = \frac{180}{\pi} \arccos\left(\frac{\mathbf{p}_1 \cdot \mathbf{p}_2}{|\mathbf{p}_1||\mathbf{p}_2|}\right)$,
- `rapidity_separation`, $\Delta y_{12} = |y_1 - y_2|$,
- `pseudorapidity_separation`, $\Delta\eta_{12} = |\eta_1 - \eta_2|$,
- `rapidity_product`, $y_1 \times y_2$.

The second cut defined in Listing 24 is a distance cut between all positrons and tagged jets in the final state of $\Delta R > 0.4$, which is an isolation criterion so that jets too close to the lepton are removed from the list of jets.

A **multi-particle** cut involves an arbitrary number of particles. In contrast to the single-particle cuts, <jet_type> and the optional parameters <jet_tag> and <jet_order> are lists of integers. If the latter two lists are not specified or shorter than the list <jet_type>, they are complemented with zero entries. The cut function depends on all specified jet entries, an example being the invariant mass of the sum of all momenta of the specified jets. The complete set of parameters for multi-particle cuts is listed in Table 20 and their actions in Table 21. For <target> $= 0$, the action can be an ordinary cut if <action> $= -1$ or a veto cut if <action> $= 1$. A <target> > 0 is only allowed for specific cases mentioned below. The parameter <n_required> is not supported for multi-particle cuts.

Available multi-particle cut types are:

<target>	<action>	**result**
0	-1	remove event if at least one specified jet pair does not pass the cut
0	1	remove event if at least one specified jet pair passes the cut (veto)
1	1	tag jet of type <jet_type_1> if it passes the cut and remove event if less than <n_required> tagged jets of type <jet_type_1> pass the cut
1	0	untag jet of type <jet_type_1> if it does not pass the cut and remove event if less than <n_required> tagged jets of type <jet_type_1> pass the cut
1	-1	remove jet of type <jet_type_1> if it does not pass the cut and remove event if less than <n_required> (tagged or untagged) jets of type <jet_type_1> pass the cut
2	1	tag jet of type <jet_type_2> if it passes the cut and remove event if less than <n_required> tagged jets of type <jet_type_2> pass the cut
2	0	untag jet of type <jet_type_2> if it does not pass the cut and remove event if less than <n_required> tagged jets of type <jet_type_2> pass the cut
2	-1	remove jet of type <jet_type_2> if it does not pass the cut and remove event if less than <n_required> (tagged or untagged) jets of type <jet_type_2> pass the cut
<target>	<order_1>	**result**
1	1	order tagged jets of type <jet_type_1> according to p_{T} and add them to the list of ordered recombinants
<target>	<order_2>	**result**
2	1	order tagged jets of type <jet_type_2> according to p_{T} and add them to the list of ordered recombinants

Table 19: Action of pair-particle cuts.

Parameter	**possible values**	**default value**	**remark**
<name>	string	none	
<jet_type>	list of jet types	none	n_{jet}
<jet_tag>	corresponding list of jet tags	[0,...,0]	$\leq n_{\mathrm{jet}}$
<jet_order>	corresponding list of jet orders	[0,...,0]	$\leq n_{\mathrm{jet}}$
<target>	$0, 1, \ldots, n_{\mathrm{jet}}$	0	> 0 only for specific cuts
<action>	$-1, 1$	-1	-1 only for <target> $= 0$
<order>	$0, 1$	0	only for specific cuts
<min_value>	real number	0.0	
<max_value>	real number	huge(0.0)	
<mid_value>	real number	0.0	only for specific cuts

Table 20: Parameters and possible values of multi-particle cuts.

<target>	<action>	**result**
0	-1	remove event if cut not passed
0	1	remove event if cut passed (veto)
$n > 0$	1	tag first n jets of list <jet_type> if the cut is passed and remove event if cut not passed

<order>	**result**
1	set order of jets tagged by this cut and append them to the list of ordered recombinants (only for specific cuts)

Table 21: Action of multi-particle cuts.

- missing_transverse_momentum, $p_{\mathrm{T,miss}} = |\sum_i \mathbf{p}_{\mathrm{T},\nu_i}|$, (sum over all neutrinos),
- missing_energy, $E_{\mathrm{miss}} = |\sum_i E_{\nu_i}|$, (sum over all neutrinos),
- invariant_mass, $M = \sqrt{(\sum_i p_i)^2}$, (sum over all specified jets),
- mt_jets_neutrinos, $M_{\mathrm{T}} = \sqrt{2 p_{\mathrm{T,jets}} p_{\mathrm{T,miss}} [1 - \cos\Delta\phi(p_{\mathrm{jets}}, p_{\mathrm{miss}})]}$, ($p_{\mathrm{jets}}$ = momentum sum of all specified jets),
- zeppenfeld_variable, $z = \dfrac{2y_1 - y_2 - y_3}{y_2 - y_3}$,
- jet_number, $N_{\mathrm{j}} = \sum_i 1$, (sum over all specified jets),
- max_invariant_mass, $M_{\max} = \max_{i_k} \sqrt{(\sum_k p_{i_k})^2}$, (maximum over all allowed jets i_k for each specified type k),
- mid_invariant_mass, $M = M_{\mathrm{mid}} \pm \min_{i_k} |\sqrt{(\sum_k p_{i_k})^2} - M_{\mathrm{mid}}|$, (minimum over all allowed jets i_k for each specified type k),
- leading_transverse_momentum, $p_{\mathrm{T,lead}} = \max_i \mathbf{p}_{\mathrm{T},i}$, (maximum over all specified jets).

The first two cuts apply to the vector sum of the transverse momenta and energies of all neutrinos, respectively. No <jet_type> input is needed, since all jets of type 6 and 21–26 are considered as neutrinos. The cut invariant_mass applies to the invariant mass of all jets of the list <jet_type>, and mt_jets_neutrinos to the transverse mass obtained from the sum of all neutrino momenta and the sum of the momenta of all specified jets. The latter quantity depends on the azimuthal angle $\Delta\phi$ between the sum of the jet momenta and the sum of the neutrino momenta. The cut zeppenfeld_variable acts on the Zeppenfeld variable (or centrality), defined via the rapidities of the three(!) listed jets, and the cut jet_number restricts the total number of jets of the specified types via the input values <min_value> and <max_value>.

Among all multi-particle cuts, the ones discussed in this paragraph support tagging and ordering. The cut max_invariant_mass acts on the maximal invariant mass calculated from all possible jets of the specified types, e.g. for <jet_type> = 1 5 on the largest invariant mass of all jet–electron pairs, while for <jet_type> = 1 1 1 on the largest invariant mass of all combinations of three QCD jets. The cut mid_invariant_mass works analogously, but applies to the invariant mass closest to the value indicated in <mid_value>. These two cuts are particularly useful for tagging, as they determine the jets that form the maximal invariant mass or invariant mass closest to some given

Parameter	possible values	default value	remark
<name>	string	none	
<jet_type_i>	list of jet types	none	n_i
<jet_tag_i>	corresponding list of jet tags	[0,...,0]	$\leq n_i$
<jet_order_i>	corresponding list of jet orders	[0,...,0]	$\leq n_i$
<parameter_i>	list of parameters for the cut	none	
<reconstructed_particle_i>	reconstructed particle name	none	
<target>	$0, 1, \ldots, n_i - 1$	1	
<action>	$-1, 0, 1$	0	
<order>	$0, 1$	0	
<n_required>	$0, 1, \ldots$	1	
<min_value>	real number	0.0	
<max_value>	real number	huge(0.0)	

Table 22: Parameters and possible values of reconstruction cuts.

value. For <target> $= n$, they tag the first n jets of the list <jet_type> that contribute to the invariant mass relevant for the cut. Note that n must be equal to or smaller than the number of specified jet types, which allows to tag all jets that form the invariant mass or only a subset. Only one jet per entry in <jet_type> is tagged. Finally, for the cut leading_transverse_momentum, the entry with the leading transverse momentum is tagged, if n is greater than or equal to the position of the corresponding jet type in the list <jet_type>. The cut acts on the transverse momentum of the jet with the highest transverse momentum among the specified jet types. For <target> > 0, only <action> $= 1$ is allowed. For <order> $= 1$, the jets tagged by this cut are appended to the existing list of ordered jets. This is also the case if the jets were already tagged but would be tagged by the multi-particle cut as well. If no jets fulfilling the tagging criterion are found, the event is cut.

A **reconstruction cut** serves to define reconstructed objects, e.g. reconstructed neutrinos or resonances, so that it does not necessarily act as a cut. If, however, values for <max_value> and <min_value> are provided, the corresponding cut acts on the first reconstructed object. The full set of parameters for a reconstruction cut is listed in Table 22. A reconstruction cut involves an arbitrary number of particles, organised in (up to two) groups. As for the multi-particle cuts, <jet_type_i> and the optional parameters <jet_tag_i> and <jet_order_i>, $i = 1, 2$, are lists of integers. If the latter two lists are not specified or shorter than the list <jet_type_i>, they are complemented with zero entries. While the last (rightmost) entry in each list, denoted as n_ith entry in the following, defines the reconstructed object(s), the other entries specify the clusters used to define the reconstructed object(s) and are selected as for the multi-particle cuts above. More precisely, the last entry of <jet_type_i> defines the jet type of the reconstructed object, which can be either an existing type or a new type in the range 30–50, the last entry of <jet_tag_i> specifies whether the reconstructed object is tagged, and the last entry of <jet_order_i> determines whether the reconstructed tagged object is ordered and thus added to the list of ordered objects. The jet order of the reconstructed object is initially equal to zero. For cases where more than one object is reconstructed, a value of the last entry of <jet_order_i> larger than zero indicates how many of them should be ordered.

The properties of the reconstructed object can be defined in two alternative ways, either with the (real) values of <parameter_i> or with the (string) values of <reconstructed_particle_i>. When indicating its name using <reconstructed_particle_i>, the reconstructed object inherits the properties of an existing particle. Otherwise, its mass and width can be defined via the first two entries of the list <parameter_i>.

The cut parameters <action> and <order> work as for the multi-particle cuts, but apply only to the clusters used for reconstruction and not to the reconstructed ones. Note also that only jets that are tagged by reconstruction cuts are ordered by them. This is also the case if the jets were already tagged but would be tagged by the reconstruction cut as well. It is worth emphasising that, as for any other cut, the clusters used for reconstruction will either be tagged or untagged by a reconstruction cut (but not necessarily left in the same tagging state as before the cut). The action of the reconstruction cuts is summarised in Table 23. Depending on the cut, one or more reconstructed objects can be present. The event is discarded, if less than <n_required> objects are reconstructed for the first group <jet_type_1>, independently of the value of the cut observable, or if the latter fulfils the cut condition (see below). Default values for the cut parameters <target>, <action>, <order>, and <n_required> are 1, 0, 0, and 1, meaning that the clusters used for reconstruction are untagged and not ordered, and at least one recombined object must be present.

Valid values for <jet_type_i> created in the reconstruction cuts

The new jet types appearing within the reconstruction cuts must correspond to existing jet types of MoCaNLO as given in Table 4 or new ones within the range 30–50.

Available reconstruction cut types are:

- single_neutrino_pt_reconstruction,
- staggered_max_pt_tagging,
- two_resonances_likelihood_reconstruction,
- resonance_mass_reconstruction,
- resonance_trivial_reconstruction,
- cluster_cloning,
- transverse_momentum_ordering.

Only the cut two_resonances_likelihood_reconstruction requires two lists of jet types, <jet_type_1> and <jet_type_2>, while all other reconstruction cuts expect one list <jet_type_1> (together with corresponding list(s) for the tags <jet_tag_i>, <jet_order_i>, <parameter_i>, and <reconstructed_particle_i>).

The cut single_neutrino_pt_reconstruction allows to reconstruct a massless object, more precisely its longitudinal momentum p_1^z, from the transverse component of the momentum p_1 of the first entry in list <jet_type_1> and the momenta p_i of the $n_1 - 2$ entries of the same list. The equation

$$m_1^2 = \left(\sum_{i=1}^{n_1-1} p_i \right)^2$$

<target>	<action>	**result on event and all but last jets specified by <jet_type_i>**
0	-1	remove event if cut not passed, no tagging/ordering
0	1	remove event if cut passed (veto), no tagging/ordering
$n > 0$	1	tag all but last jets of list(s) <jet_type_i> with $i \leq n$ if the cut is passed and remove event if the cut is not passed
$n > 0$	0	untag all but last jets of list(s) <jet_type_i> with $i \leq n$ if the cut is passed and remove event if the cut is not passed
<order>		**result on all but last jets specified by <jet_type_i>**
1		set order of jets tagged by this cut and append them to the list of ordered recombinants, if <target> > 0
<jet_type_i>last		**result on last jet specified by <jet_type_i>**, if <target> > 0
n		set type of reconstructed object to n (use an existing particle type or a new one between 30 and 50)
<jet_tag_i>last		**result on last jet specified by <jet_type_i>**, if <target> > 0
1/0		tag/do not tag reconstructed object
<jet_order_i>last		**result on last jet specified by <jet_type_i>**, if <target> > 0
1/0		order/do not order reconstructed objects, i.e. add them to the list of tagged jets
$n > 0$		generate new list of n ordered objects,
		only for cut `transverse_momentum_ordering`
<n_required>		**result on last jet specified by <jet_type_1>**
n		require at least n reconstructed objects for the first group
		(not necessarily passing the cut condition)

Table 23: Action of reconstruction cuts. Only the listed combinations of <target> and <action> are allowed.

is solved for the unknown p_1^z, where m_1 is the mass of the cluster specified via the first entry of <parameter_1> or alternatively via <reconstructed_particle_1>. The reconstructed object, with the reconstructed longitudinal momentum p_1^z, is put in the n_1th entry of <jet_type_1>. If two solutions exist, reconstructed objects are generated for both, the first entry being the one with smaller longitudinal momentum. If the solutions are complex, the real part is selected. No cut is applied if at least one reconstructed neutrino is found.

The cut `staggered_max_pt_tagging` creates a new jet type with exactly one jet in its recombinant list from the list of all jet types specified by the first $n_1 - 1$ entries of <jet_type_1>. The new object is the one with largest transverse momentum of the first type in <jet_type_1>. If this list is empty, the one with largest transverse momentum of the second type in <jet_type_1> is searched. This is continued until one object is found or all $n_1 - 1$ types have been searched. The cut acts on the transverse momentum of the resulting object.

The cut `two_resonances_likelihood_reconstruction` reconstructs two resonances from the objects in the first n_1-1 and n_2-1 entries of the lists `<jet_type_1>` and `<jet_type_2>`, respectively. From all allowed clusters with fitting tags and orders, those objects are selected that maximise the likelihood

$$\mathcal{L} = \frac{1}{(M_1^2 - m_1^2)^2 + (m_1\Gamma_1)^2} \, \frac{1}{(M_2^2 - m_2^2)^2 + (m_2\Gamma_2)^2} \,,$$

where

$$M_1^2 = \left(\sum_{i_1=1}^{n_1-1} p_{i_1} \right)^2 , \qquad M_2^2 = \left(\sum_{i_1=1}^{n_2-1} p_{i_2} \right)^2 ,$$

and m_i and Γ_i are the masses and widths of the two resonances specified via the first two entries of `<parameter_i>` or via `<reconstructed_particle_i>`. For `<target>` $= 1$, the tagging applies to all clusters that reconstruct the first resonance, for `<target>` $= 2$, it applies to all clusters that reconstruct the two resonances. The tagging of the resonances is steered by the values of the last entries in `<jet_tag_i>`. A possible cut acts on the invariant mass of the reconstructed resonance in the first list `<jet_type_1>`.

The cuts `resonance_mass2_reconstruction` and `resonance_mass_reconstruction` reconstruct a resonance from the objects in the first $n_1 - 1$ entries of the list `<jet_type_1>`. From all allowed clusters with fitting tags and orders, those objects are selected that minimise the distances

$$\Delta = |M_1^2 - m_1^2| \quad \text{or} \quad \Delta = |M_1 - m_1|,$$

respectively, where

$$M_1^2 = \left(\sum_{i_1=1}^{n_1-1} p_{i_1} \right)^2 ,$$

and m_1 is the mass of the resonance specified via the first entry of `<parameter_1>` or via `<reconstructed_particle_1>`. Tagging applies to all clusters that reconstruct the resonance. A possible cut acts on the invariant mass of the reconstructed resonance upon setting the `<min_value>` and/or `<max_value>`.

The cut `resonance_trivial_reconstruction` reconstructs an object from those in the first n_1-1 entries of the list `<jet_type_1>`. For each entry, the first allowed cluster with fitting tags and orders is taken. Tagging applies to all clusters that reconstruct the resonance. A possible cut acts on the invariant mass of the reconstructed resonance.

The cut `cluster_cloning` clones the objects in the first $n_1 - 1$ entries of the list `<jet_type_1>` into the new jet type. For each entry, all allowed clusters with fitting tags and orders are taken. No cut is applied. This cut does not change the tagging status of the input jets as opposed to the other reconstruction cuts. It can be used to combine objects with different jet types into one new jet type to be used in subsequent cuts, histograms or scales.

The cut `transverse_momentum_ordering` serves to order objects of different jet types according to their transverse momenta. Up to the number of objects specified by the n_1th entry of `<jet_order_1>` fitting to the first $n_1 - 1$ entries of `<jet_type_1>` are copied into a list for a new jet type (defined by this cut) and ordered by their transverse momenta. This can for instance be used to order leptons of different types (electrons, positrons, muons, ...) according to their transverse momenta. A possible cut acts on the transverse momentum of the selected object with largest transverse momentum. This cut does not change the tagging of the input jets.

```
<cuts id="tzj">
    <cut type="two_resonances_likelihood_reconstruction ">
        <name>                top_Z_tagging </name>
        <jet_type_1>          16 6 2 7      </jet_type_1>
        <jet_tag_1>           0  0 0 1      </jet_tag_1>
        <jet_order_1>         0  0 0 0      </jet_order_1>
        <reconstructed_particle_1> t </reconstructed_particle_1>
        <jet_type_2>          4 5 8         </jet_type_2>
        <jet_tag_2>           0 0 1         </jet_tag_2>
        <jet_order_2>         0 0 0         </jet_order_2>
        <reconstructed_particle_2> z </reconstructed_particle_2>
        <n_required>          1             </n_required>
        <target>              2             </target>
        <action>              1             </action>
        <order>               1             </order>
    </cut>
    ...
    <cut type="transverse_momentum_ordering">
        <name>              lepton_pt_sort </name>
        <jet_type_1>          4  5 16 32    </jet_type_1>
        <jet_tag_1>           1  1  1  1    </jet_tag_1>
        <jet_order_1>         1  1  1  3    </jet_order_1>
        <n_required>          1             </n_required>
        <target>              0             </target>
        <order>               0             </order>
    </cut>
</cuts>
```

Listing 25: Examples for reconstruction cuts.

Examples for the use of reconstruction cuts are provided in Listing 25. The first cut, named `top_Z_tagging`, reconstructs a top quark (`<jet_type>` = 7) from all untagged positive muons (16), neutrinos (6), and bottom jets (2), and a Z boson (8) from all untagged electrons (4) and positrons (5), and tags and orders all objects that reconstruct the top quark and the Z boson. The parameters for the masses and widths of the top quark and Z boson are taken from the values for the corresponding particles. The second cut, named `lepton_pt_sort`, sorts all tagged electrons, positrons and positively charged muons with `<jet_order>` equal to 1 according to their transverse momenta and puts the first three objects into the `<jet_type>` 32.

The typical use of the cut scheme for a VBS process with jet tagging can be found in Listing 24. The first cut of type `transverse_momentum` acts on all jets (`<jet_type>` = 1) and tags them if their transverse momentum is above 30 GeV. The following cut of type `distance` acts on tagged jets and untags them if their distance to positrons (`<jet_type>` = 4) is below 0.4. The third cut of type `rapidity` acts on tagged jets, untags them if their rapidity is outside the interval $[-4.7, 4.7]$, and orders the tagged jets. Finally, the cut of type `invariant_mass` acts on tagged jets with specified order, i.e. the two leading transverse-momentum jets, and discards events with $M_{\mathrm{j_1 j_2}} < 400\,\mathrm{GeV}$. Note that the order of the cuts is crucial.

7.4. Fixing scales via the cut card

Dynamical scales can be defined via the cut card (`cut_card.xml`) by using generic scale types generated from selected final-state jets. To this end, the tag `<dynamical_scale_type>` (in the `<run_parameters>` section of the `param_card.xml`) must be set to 1 and the definition of scales must be provided in the cut card. This requires a corresponding tag `<scales id="...">` in `run_card.xml`. If this tag is not found, the code will stop with an error message.

observable	type
$(\prod_{i=1}^{n} p_{\mathrm{T},i})^{1/n}$	`transverse_momentum_product`
$(\sum_{i=1}^{n} p_{\mathrm{T},i}^{m})^{1/m}$	`transverse_momentum_sum`
$\max_i p_{\mathrm{T},i}$	`transverse_momentum_maximum`
$(\prod_{i=1}^{n} E_{\mathrm{T},i})^{1/n}$	`transverse_energy_product`
$(\sum_{i=1}^{n} E_{\mathrm{T},i}^{m})^{1/m}$	`transverse_energy_sum`
$\max_i E_{\mathrm{T},i}$	`transverse_energy_maximum`
$(\prod_{i=1}^{n} M_{\mathrm{T},i})^{1/n}$	`transverse_mass_product`
$(\sum_{i=1}^{n} M_{\mathrm{T},i}^{m})^{1/m}$	`transverse_mass_sum`
$\max_i M_{\mathrm{T},i}$	`transverse_mass_maximum`
$(\prod_{i=1}^{n} M_i)^{1/n}$	`invariant_mass_product`
$(\sum_{i=1}^{n} M_i^m)^{1/m}$	`invariant_mass_sum`
$\max_i M_i$	`invariant_mass_maximum`
$(\prod_{i=1}^{n} m_i)^{1/n}$	`particle_mass_product`
$(\sum_{i=1}^{n} m_i^m)^{1/m}$	`particle_mass_sum`

Table 24: Implemented individual scale types. The final state jets i are selected via the input tags `<jet_type>`, `<jet_tag>`, and `<jet_order>`. The `<power>` of the scale is denoted my m.

Parameter	possible values	default value
`<name>`	string	none
`<jet_type>`	list of jet types	none
`<jet_tag>`	corresponding list of jet tags	none
`<jet_order>`	corresponding list of jet orders	none
`<jet_mass>`	real number	0.0
`<jet_particle>`	particle name	none
`<offset>`	real number	0.0
`<factor>`	real number	1.0
`<power>`	real number	1.0
`<weight>`	real number	1.0

Table 25: Parameters of an individual scale type (some parameters are not available for all types).

The central scale can be constructed by multiplicatively combining several scale sums l, which in turn are sums of individual scales k. The implemented types for individual scales are listed in Table 24 and the corresponding parameters in Table 25. We define the transverse energy for a momentum p with transverse part p_{T} via

$$E_{\mathrm{T}} = \sqrt{p^2 + p_{\mathrm{T}}^2} \tag{5}$$

Parameter	possible values	default value
<power>	real number	1
<weight>	real number	1

Table 26: Parameters of a scale sum.

and the transverse mass via

$$M_{\mathrm{T}} = \sqrt{m^2 + p_{\mathrm{T}}^2}\,, \tag{6}$$

where m is a mass associated to the corresponding jet (see below). The individual scales are calculated by taking the geometric average, the sum, or the maximum of the corresponding observable for a selection of final-state objects specified via the tags <jet_type>, <jet_tag>, and <jet_order> > 0. Note that the jets are selected according to the relevant tagging scheme as described at the end of Sec. 8.

The jets can be associated a mass indicated via the tag <jet_mass> or the mass of the particle specified via the tag <jet_particle>, which can take the values in column 2 of Table 4. In addition, the resulting quantity can be increased by an offset <offset> (typically obtained from the masses of certain particles) and multiplied by a factor <factor>. An individual scale of type *xxx*_product, where xxx refers to the observables in Table 24, is calculated via

$$scale_k = factor_k * \left(\prod_{i=1}^{n} (observable_{k,i})^{1/n} + offset_k \right)$$

and an individual scale of type *xxx*_sum via

$$scale_k = factor_k * \left[\left(\sum_{i=1}^{n} (observable_{k,i})^{power_k} \right)^{1/power_k} + offset_k \right],$$

where *power* is fixed by the tag <power>, and an individual scale of type *xxx*_max via

$$scale_k = factor_k * (\max_i observable_{k,i} + offset_k).$$

Individual scales can be combined in a scale sum indicated by the tag <scale_sum> as

$$scale\_sum_l = \left[\sum_k (scale_k)^{power_l} \right]^{1/power_l},$$

where <power> is the parameter of the scale sum l (see Table 26). A scale sum can receive a further parameter <weight>, which is relevant for the calculation of the final scale according to Eq. (7). A <scale> that is not part of a <scale_sum> environment is considered as a scale sum with just this scale. The <weight> of this scale is passed to the corresponding <scale_sum>. The tag <weight> of the individual scale is ignored for a <scale> within a <scale_sum>.

The final scale is combined from all scale sums defined in the input card via

$$final\_scale = global\_factor * \left[\prod_l scale\_sum_l^{weight_l} \right]^{\frac{1}{\sum_l weight_l}}, \tag{7}$$

```
<scales id="ttxbbx">
    <global_factor>  .5d0       </global_factor>
    <scale_sum>
    <weight>          1.0       </weight>
    <scale type="transverse_energy_sum">
        <name>  transverse_energy_bottom_pair </name>
        <jet_type>      2       </jet_type>
        <jet_tag>       0       </jet_tag>
        <jet_order>     0       </jet_order>
        <factor>        1       </factor>
    </scale>
    </scale_sum>
    <scale_sum>
    <weight>          1.0       </weight>
    <scale type="transverse_momentum_sum">
        <name>  missing_transverse_momentum </name>
        <jet_type>     31       </jet_type>
        <jet_tag>       1       </jet_tag>
        <jet_order>     1       </jet_order>
        <factor>        1       </factor>
    </scale>
    <scale type="transverse_energy_sum">
        <name>  transverse_energy_charged_fermions </name>
        <jet_type>      4 17 2  </jet_type>
        <jet_tag>       0 0  0  </jet_tag>
        <jet_order>     0 0  0  </jet_order>
        <factor>        1       </factor>
        <offset>      346       </offset>
    </scale>
    </scale_sum>
</scales>
```

Listing 26: Example for scale definition via `cut_card.xml`.

where $global\_factor$ is the value of the tag `<global_factor>` of the complete list of scales.

If `<global_factor>` and the `<weight>` of all `<scale_sum>` subsections contain two entries each, the first set of entries is used to construct the factorisation scale and the second one the renormalisation scale. Thus, `<scale_sum>` sections with `<weight>` w1 0 `</weight>` define the factorisation scale, while the ones with `<weight>` 0 w2 `</weight>` define the renormalisation scale. Sections with `<weight>` w1 w2 `</weight>` enter the definition of both scales.

An example for the definition of a scale used for $pp \to t\bar{t}b\bar{b} \to \mu^- \bar{\nu}_\mu e^+ \nu_e \bar{b} b \bar{b} b$ is given in Listing 26. This defines the scale

$$\mu_0 = \frac{1}{2}\left[\left(\sum_{i=j_b} E_{T,i}\right)\left(p_T^{miss} + \sum_{i=e^+,\mu^-,j_b} E_{T,i} + 346\,\text{GeV}\right)\right]^{1/2}, \tag{8}$$

where j_b denotes all bottom and antibottom jets. The missing transverse momentum p_T^{miss} is defined via the reconstruction cut in Listing 27. The top-quark mass is accounted for by the offset $2m_t = 346\,\text{GeV}$. Alternatively, it can be taken into account by adding an extra scale as shown in Listing 28 to the second `<scale_sum>` in Listing 26.

Another example used for $pp \to e^+e^-\gamma$ is shown in Listing 29 and defines the scale

$$\mu_0 = \frac{1}{\sqrt{2}}\left[M_Z^2 + \sum_{i=Z,\gamma,j} p_{T,i}^2\right]^{1/2}. \tag{9}$$

```
<cut type="resonance_trivial_reconstruction">
    <name> missing_momentum </name>-
    <jet_type_1>    6  6 31 </jet_type_1>
    <jet_tag_1>     0  0  1 </jet_tag_1>
    <jet_order_1>   0  0  1 </jet_order_1>
    <target>        1       </target>
    <action>        0       </action>
    <order>         0       </order>
</cut>
```

Listing 27: Definition of missing momentum via a reconstruction cut in `cut_card.xml`.

```
<scale type="particle_mass_sum">
    <name>  particle_mass_top </name>
    <jet_type>      7      </jet_type>
    <jet_tag>       1      </jet_tag>
    <jet_order>     1      </jet_order>
    <jet_particle>  t      </jet_particle>
    <factor>        2      </factor>
</scale>
```

Listing 28: Definition of scale from mass of top particle times 2.

7.5. Consistency checks of input

MoCaNLO performs some consistency checks on the input. When reading the cuts and scales, it checks, in particular, whether the input for `<jet_type>`, `<jet_tag>`, and `<jet_order>` matches with the final state of the considered process or the objects constructed during the recombination and cut procedure. By default, the code stops if it detects a potential inconsistency, which is useful to correctly set up the input cards. At their own risk, the stop can be switched off for some warnings by the user by adding

```
<ignore_stop>  true </ignore_stop>
```

in the `<cut>` or `<scale>` environment. Moreover, by using the tag

```
<ignore_input_check>  true </ignore_input_check>
```

all checks for a particular `<cut>` or `<scale>` can be disabled. However, this is not recommended, as it usually leads to run-time errors.

8. Listing observables with the `plot_card.xml`

The plot card `plot_card.xml` contains the section `<histograms>`, where histograms of observables are listed. Individual histograms are defined in `<histogram>` sections, with their type specified by the attribute `type`. The histograms operate on momenta after cuts, i.e. on recombined and/or reconstructed objects called generically jets.

The implemented histogram types are listed in Table 27. The parameters of a histogram are summarised in Table 28. Listing 30 shows as an example the definition of a histogram of the transverse momentum of a bottom-jet pair. The parameter `<name>` is used for naming the resulting data file, e.g. for the example in Listing 30 it reads `histogram_transverse_momentum_bb12.dat`. The tags `<jet_type>`, `<jet_tag>`, and `<jet_order>` indicate which transverse momentum to plot. The arrays of integers in `<jet_type>` specify the types of jets, those in `<jet_tag>` their tagging status and those in `<jet_order>` the position of the jets in the ordered lists for `<jet_type>` as defined by

category	observable	type
single-jet based	$\sum_n E_n$	`energy`
single-jet based	$(\sum_n p_n)_\mathrm{T}$	`transverse_momentum`
single-jet based	$\sum_n \lvert p_{\mathrm{T}_n}\rvert$	`transverse_momentum_scalar_sum`
single-jet based	$\sum_n E_{\mathrm{T}n}$	`transverse_energy`
single-jet based	$\sum_n M_{\mathrm{T}n}$	`transverse_mass`
single-jet based	$\sqrt{(\sum_n p_n)^2}$	`invariant_mass`
single-jet based	$y(\sum_n p_n)$	`rapidity`
single-jet based	$\eta(\sum_n p_n)$	`pseudorapidity`
single-jet based	$\cos\theta(\sum_n p_n)$	`cosine_angle`
single-jet based	$\tilde{M}_\mathrm{T}$	`transverse_mass_neutrino`
multi-jet based	ΔR_{ij}	`distance`
multi-jet based	Δr_{ij}	`pseudodistance`
multi-jet based	ϕ_{ij}	`azimuthal_angle_separation`
multi-jet based	$\cos\theta_{ij,k}$	`cosine_angle_separation`
multi-jet based	y_{ij}	`rapidity_separation`
multi-jet based	η_{ij}	`pseudorapidity_separation`
multi-jet based	$p_{\mathrm{T},j}/p_{\mathrm{T},i}$	`transverse_momentum_ratio`
multi-jet based	$\cos\theta^*_{ij,k}$	`cosine_decay_angle`
multi-jet based	$\phi^*_{ijkl,m}$	`azimuthal_angle_planes`
multi-jet based	z_{ijk}	`z3rap`
overall	H_T	`total_transverse_energy`
overall	$\sqrt{\hat{s}}$	`cms_energy`

Table 27: Implemented histogram types.

```
<histogram type="transverse_momentum">
    <name>              "bb12"                                  </name>
    <jet_type>          2 2                                     </jet_type>
    <jet_tag>           1 1                                     </jet_tag>
    <jet_order>         1 2                                     </jet_order>
    <x_min>             0d0                                     </x_min>
    <x_max>             13000.0                                 </x_max>
    <bin_width>         20.0                                    </bin_width>
    <latex_title>       "Transverse Momentum b-Jet Pair"        </latex_title>
    <latex_observable>  "p_{\text{T},\text{b}_1\text{b}_2}"     </latex_observable>
    <plot_xmin>         0.0                                     </plot_xmin>
    <plot_xmax>         400.0                                   </plot_xmax>
    <plot_ymin>         1d-4                                    </plot_ymin>
    <plot_ymax>         0.01                                    </plot_ymax>
    <plot_kmin>         0.6                                     </plot_kmin>
    <plot_kmax>         2.2                                     </plot_kmax>
    <logscale>          true                                    </logscale>
</histogram>
```

Listing 30: Transverse-momentum histogram in `plot_card.xml`.

```
<scales id="eexa">
    <global_factor> 0.7071067811865475d0 </global_factor>
    <scale_sum>
        <weight> 1.0 </weight>
        <power> 2.0 </power>
        <scale type="transverse_momentum_sum">
            <name> transverse_momenta_squared </name>
            <jet_type>     23 3 1  </jet_type>
            <jet_tag>       0 0 0  </jet_tag>
            <jet_order>     0 0 0  </jet_order>
            <factor>        1      </factor>
            <power>         2      </power>
        </scale>
        <scale type="particle_mass_sum">
            <name> particle_mass_squared </name>
            <jet_type>     23      </jet_type>
            <jet_tag>       0      </jet_tag>
            <jet_order>     0      </jet_order>
            <jet_particle>  z      </jet_particle>
            <factor>        1      </factor>
        </scale>
    </scale_sum>
</scales>
```

Listing 29: Example for scale definition via `cut_card.xml`.

Parameter	possible values	default value
`<name>`	string	none
`<jet_type>`	list of jet types	none
`<jet_tag>`	corresponding list of jet tags	[0,...,0]
`<jet_order>`	corresponding list of jet orders	[0,...,0]
`<x_min>`	real number	histogram-specific
`<x_max>`	real number	histogram-specific
`<bin_width>`	real number	histogram-specific
`<x_scale_up>`	real number	1.0
`<x_scale_down>`	real number	1.0
`<abs>`	true/false	false
`<use_tagging>`	0,1,2	1
`<latex_title>`	string	none
`<latex_observable>`	latex_string	none
`<plot_xmax>`	real number	`<x_max>`
`<plot_xmin>`	real number	`<x_min>`
`<plot_ymax>`	real number	none
`<plot_ymin>`	real number	none
`<plot_kmax>`	real number	none
`<plot_kmin>`	real number	none
`<logscale>`	true/false	histogram-specific

Table 28: Parameters of a histogram.

the tagging scheme (see below). While <jet_type> and <jet_order> are mandatory for all jet-based histograms, <jet_tag> is assumed to be zero if not present. The definitions in Listing 30 correspond to add the momenta of the first and second tagged and ordered b jets and to bin the transverse momentum of this sum, labelled as $p_{\mathrm{T},\mathrm{b}_1\mathrm{b}_2}$. The tags <x_min> and <x_max> fix the observable's range to bin and <bin_width> the bin width to be used.

The tags <x_scale_up>, <x_scale_down>, and <abs> are optional and modify the default settings of the histogram. If <abs> is set to true, the absolute value of the observable is binned. With <x_scale_up> and <x_scale_down>, the observable is multiplied and divided by the respective input number before filling the bins. By default, observables with mass dimension are measured in GeV and angles in degrees. In the latter case, setting <x_scale_down> to 180 and <x_scale_up> to 3.1415 will display the distributions in radians.

The remaining tags are only passed through the histogram files to gnuplot. The strings given in <latex_title> and <latex_observable> are used as title and observable label of the histogram in the LaTeX output. The other tags define the visible plot ranges for the x axis via <plot_xmin> and <plot_xmax>, the ranges of the y axis via <plot_ymin> and <plot_ymax>, while <plot_kmin> and <plot_kmax> define the y axis range of the k factor in relative plots. The tag <logscale> allows to specify whether to use a logarithmic scale in the plot.

For the single-jet-based histogram types, the lists <jet_type>, <jet_tag>, and <jet_order> can have an arbitrary number of entries (same number for all lists) and the observables of all partons that match the corresponding entries are summed up as indicated in Table 27. The transverse energy E_{T} and the transverse mass M_{T} are defined as in Eq. (5) and Eq. (6), respectively. In the histogram of type transverse_mass_neutrino, the momenta of jets of type neutrino are summed separately from all other entries, resulting in two momenta p_ν and p_{vis}. From these momenta the transverse mass is calculated as

$$\tilde{M}_{\mathrm{T}} = \sqrt{(p_{\mathrm{T,vis}} + p_{\mathrm{T},\nu})^2 - (\mathbf{p}_{\mathrm{T,vis}} + \mathbf{p}_{\mathrm{T},\nu})^2}.$$

For multi-jet-based histogram types with two indices ij, exactly two entries are expected in each of these lists. In those cases all observables are computed from momenta in the laboratory system. For the observables with indices ij, k either two or three entries can be given. If the third entry is missing for cosine_angle_separation, $\cos\theta_{ij}$ is calculated in the laboratory system. If the third entry is missing for cosine_decay_angle, $\cos\theta^*_{ij}$, i.e. the decay angle of parton i in the rest frame of parton j, is computed by first boosting the momentum i in the frame of j, and evaluating in this frame the angle of i with respect to the boost direction. If the third argument k is present, in both cases all momenta are first boosted to the rest frame of parton k, and thereafter the angles are calculated as specified above. The histogram azimuthal_angle_planes works similarly and produces a distribution in the angle between two planes, defined by the momenta of the first two pairs of the specified jets. Without a fifth jet specified, this angle is calculated in the laboratory frame, otherwise it is calculated in the rest system of the fifth jet. Finally, the Zeppenfeld variable (or centrality) is defined as

$$z_{ijk} = \frac{2y_i - y_j - y_k}{2(y_j - y_k)}$$

and requires exactly three entries.

For the histograms total_transverse_energy and cms_energy the tags <jet_type>, <jet_tag>, and <jet_order> are not allowed. The former histogram yields the sum of the transverse energies of all final-state objects, including neutrinos (missing energy) and bremsstrahlung objects. The histogram cms_energy yields the total partonic CM energy.

Histograms can be based on three different tagging schemes, which are selected using the tag <use_tagging>, which either applies to all histograms (when specified in the section <histograms>) or to individual histograms (when specified within the <histogram> definitions). The default is <use_tagging> = 1.

For <use_tagging> = 0, tagging is not used, i.e. <jet_tag> should not be present. All jets are taken from the lists of *recombinants* of the corresponding <jet_type>, and <jet_order> refers to the corresponding position in this list. The ordering in the list of *recombinants* corresponds to an ordering in transverse momenta.

Use of <use_tagging> = 0

The tag <jet_order> refers to the position in the list of recombinants, regardless of whether some of the recombinants have been cut. Thus, this option should only be used for simple cases and if no tagging is required.

For <use_tagging> = 1 (default), the tagging information of the cut routines is consistently used. Specific jets *must* be tagged and ordered and only ordered jets can be addressed via the tag <jet_order>. *This might require the use of extra cuts that take care of the tagging.* For <jet_order> > 0, the jets are taken from the list of *ordered recombinants*, and <jet_order> refers to the corresponding position in this list. For <jet_order> = 0, the jets are taken from the list of *recombinants* after discarding those entries whose tag does not match <jet_tag> or with <jet_order> ≠ 0. For histograms in the single-jet category, all matching jets are taken into account with <jet_order> = 0. This is useful if extra NLO jets, i.e. jets that may or may not be present in the final state, are to be included. For histograms in the multi-jet category, <jet_order> = 0 is not allowed.

For <use_tagging> = 2, the tagging information of the cut routines is used in a different way. If <jet_tag> is missing in the histogram definition, the jets are assumed not to be tagged. Furthermore, all tagged jets must also be ordered. For <jet_tag> > 0, the jets are taken from the list of *ordered recombinants*. For <jet_tag> = 0, the jets are taken from the list of *recombinants* after discarding those entries with non-zero <jet_tag>. In both cases, <jet_order> refers to the corresponding entry in this list and not necessarily to the corresponding <jet_order> as defined by the cuts. For histograms that can take lists of jets, all matching jets are taken into account. This scheme allows to access untagged jets in *recombinants* via <jet_order>.

A further example is shown in Listing 31, which is understood together with the cuts defined in Listing 24, which define the two tagged jets of the process according to the jet_rapidity cut. The histogram bins the invariant mass of the two tagged jets, calculated from the first and second momenta (<jet_order> = 1 2) of the ordered recombinants.

As for cuts and scales, MoCaNLO checks the input for <jet_type>, <jet_tag>, and <jet_order> also for histograms. By setting the tag

```
<ignore_stop>  true </ignore_stop>
```

in the <histogram> environment, one can prevent the code from stopping, and if one sets

```
<ignore_input_check>  true </ignore_input_check>
```

all checks are switched off for the corresponding histogram. This, however, might lead to run-time errors.

```
<histogram type="invariant_mass">
    <name>              tag_jets                        </name>
    <jet_type>          1 1                             </jet_type>
    <jet_order>         1 2                             </jet_order>
    <jet_tag>           1 1                             </jet_tag>
    <x_min>             500                             </x_min>
    <x_max>             4000                            </x_max>
    <bin_width>         50                              </bin_width>
    <latex_title>       Invariant mass tag jets         </latex_title>
    <latex_observable>  M_{\mathrm{j}_1\mathrm{j}_2}    </latex_observable>
    <plot_xmin>         500                             </plot_xmin>
    <plot_xmax>         4000                            </plot_xmax>
    <plot_ymin>         0.00001                         </plot_ymin>
    <plot_ymax>         0.1                             </plot_ymax>
    <plot_kmin>         0.5                             </plot_kmin>
    <plot_kmax>         1.6                             </plot_kmax>
    <logscale>          true                            </logscale>
</histogram>
```

Listing 31: Histogram for the invariant mass of the two tagged jets, which are defined by the `invariant_mass_tag_jets` cut of Listing 24.

9. User-defined input features

Despite the general structure and flexibility of the cards, there might still be cuts, scales, or histograms that cannot be obtained from the input in the cards. These objects can be defined in dedicated Fortran files, which are named `user_defined_cuts.F90`, `user_defined_scales.F90`, and `user_defined_histograms.F90`, respectively.

Templates for such files can be found in the folder

```
/path/to/mocanlo/src/mocanlo/user_defined_inputs
```

Depending on the user's needs, one, two, or all of them must be copied from that location to the folder of the process to be computed, and specifically in a subfolder named `user_defined`. As an example, we consider the process $pp \to e^+e^-\mu^-\bar{\nu}_\mu jj$ with polarised intermediate bosons, to be found in the folder `/path/to/process/vbs/wz/pp_wz_ew_ew_pol`. Once the necessary files have been copied, the process directory will look like

```
pp_wz_ew_ew_pol/
├── cards/
│   └── ...
└── user_defined/
    ├── user_defined_cuts.F90
    ├── user_defined_scales.F90
    └── user_defined_histograms.F90
```

Then, the files must be modified accordingly, as described below. Finally, to link these files when compiling the code, the script `compile_mocanlo` must be run with the (relative or absolute) path of the process folder as an argument, like

```
./compile_mocanlo /path/to/processes/vbs/wz/pp_wz_ew_ew_pol
```

The modification of a user-defined file always requires three steps: specifying the numbers of new features that one wants to implement, setting a template for all of them, and setting up the procedure that contains their actual definitions.

We start showing how a new cut acting on a Zeppenfeld-like variable can be defined for pp $\to$ $e^+e^-\mu^-\bar{\nu}_\mu jj$ by modifying the file `user_defined_cuts.F90`. The content of the file is reported in Listing 32.

```
#include "helpers.h"

module user_defined_cuts
  use user_defined_cut2
  implicit none

  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (1) Set number of user-defined cuts
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  integer :: n_user_cuts = 1

  type, extends(mc_user_defined_cut) :: mc_zep_cut
  contains
    procedure :: calc_observable => mc_zep_cut_get_value
  end type mc_zep_cut

contains


  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (2) Add template to list
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  subroutine init_user_cuts_templ(user_template,index)
    class(mc_user_defined_cut), pointer :: user_template
    integer :: index

    if(index==1)then
      allocate(mc_zep_cut :: user_template)
      call user_template%init_template(label="zep_cut") ! <- This is the cut type
    end if

  end subroutine init_user_cuts_templ

  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (3) Define new cuts
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!
  function mc_zep_cut_get_value(this, particles, sorted_clusters2) result(cut_applied)
    class(mc_zep_cut), intent(in) :: this
    logical :: cut_applied
    class(mc_particle), dimension(:), intent(inout) :: particles
    type(mc_recombinant2_list), dimension(:), intent(inout), target :: sorted_clusters2
    real(kind=dp) :: zeppenfeld_variable
    real(kind=dp) :: lep_rap, j1_rap, j2_rap
    integer :: jet_type, lep_type, i, n_tagged_jets, i_order
    logical :: ordered_jets

    ignore(particles) ! Act on sorted clusters, not at truth level
    cut_applied = .false.

    ! This cut is applied on a Zeppenfeld variable. This cut only operates
    ! on tagged jets that must be ordered. If less than 2 tagged jets are found,
    ! the cut is applied. If at least 2 tagged jets are found, but they are not ordered,
    ! this cut orders them and puts them in the list of ordered_recombinants.
    ! Only at this point the observable is computed.

    ! (1) Check if at least two tagged light jets exist
    jet_type = 1 ! Light jets
    n_tagged_jets = 0
    ordered_jets = .true.
    do i = 1, sorted_clusters2(jet_type)%n_recombinants
      if (sorted_clusters2(jet_type)%recombinants(i)%ptr%get_jet_tag() == 1) then
        n_tagged_jets = n_tagged_jets + 1
        if(sorted_clusters2(jet_type)%recombinants(i)%ptr%get_jet_order() == 0) then
          ordered_jets = .false.
```

```
      end if
    end if
  end do
  if(n_tagged_jets < 2) then
    cut_applied = .true.
    return
  end if

  ! (2) If needed, order the tagged jets and add them to list of ordered recombinants
  i_order = 0
  if(.not.ordered_jets) then
    do i = 1, sorted_clusters2(jet_type)%n_recombinants
      if (sorted_clusters2(jet_type)%recombinants(i)%ptr%get_jet_tag() == 1) then
        i_order = i_order + 1
        sorted_clusters2(jet_type)%n_ordered_recombinants = i_order
        ! Only the variable 'order' of the recombinants is set, since in this
        ! example we assume that the objects in the sorted_clusters2 list are
        ! already ordered in their transverse momentum
        call sorted_clusters2(jet_type)%recombinants(i)%ptr%set_jet_order(i_order)
        sorted_clusters2(jet_type)%ordered_recombinants(i_order) = &
          sorted_clusters2(jet_type)%recombinants(i)
      else
        call sorted_clusters2(jet_type)%recombinants(i)%ptr%set_jet_order(0)
      end if
    end do
  end if

  ! (3) Compute variable
  lep_type = 4 ! Positron
  lep_rap = sorted_clusters2(lep_type)%recombinants(1)%ptr%rapidity()
  j1_rap = sorted_clusters2(jet_type)%ordered_recombinants(1)%ptr%rapidity()
  j2_rap = sorted_clusters2(jet_type)%ordered_recombinants(2)%ptr%rapidity()

  zeppenfeld_variable = ( lep_rap - 0.5 * (j1_rap + j2_rap) ) / abs(j1_rap - j2_rap)

  if(abs(zeppenfeld_variable) > 0.4) cut_applied = .true.

end function mc_zep_cut_get_value

end module user_defined_cuts
```

Listing 32: Example of a `user_defined_cuts.F90` file for the process $pp \to e^+e^-\mu^-\bar{\nu}_\mu jj$.

First, the variable `n_user_cuts` is set to one, i.e. the number of new cuts that we want to add. Right below, a new type `mc_zep_cut` is declared, inheriting from the type `mc_user_defined_cut`. The new type just needs to define its own `calc_observable` procedure, which in the example here is renamed `mc_zep_cut_get_value`.

As a second step, the if-statement in the subroutine `init_user_cuts_templ` has to be adapted. Specifically, for each new cut type an else-if scope must be added with the condition `index==n`, where `n` is an integer number indexing the different cuts. In the corresponding scope, a template for the cut has to be allocated and initialised. For a user-defined cut, this just requires to set a label. This label defines the type with which the cut becomes accessible from the `cut_card.xml`:

```
<cut type="zep_cut">
    <name>  my_zep_cut </name>
</cut>
```

Note that, as for any other cut, the position of the user-defined cut in the sequence of cuts in the `cut_card.xml` is important.

Finally, the function `mc_zep_cut_get_value` must be defined. The function is expected to return a logical variable `cut_applied` that tells whether the cut must be applied or not. Moreover, it always operates on `sorted_clusters2`, which collects the reconstructed objects for all jet types

possibly resulting from the recombination chain and the action of previous cuts. Each of these jet types is a component of the `sorted_clusters2` array. Depending on the process, each component can collect multiple objects of the same type. Such objects are accessible via two different lists (for further context, see discussion at the beginning of Sec. 7.3):

- **`sorted_clusters2(jet_type)%recombinants(i)%ptr`** is the list of *recombinants* containing `sorted_clusters2(jet_type)%n_recombinants` objects of type `jet_type`. This is the only way of accessing objects if they have not been ordered by the action of a cut via the tag `<order>`. One can inquire whether an object in the list has been tagged or ordered by checking the value of `get_jet_tag()` and `get_jet_order()`, respectively.
- **`sorted_clusters2(jet_type)%ordered_recombinants(i)%ptr`** is the list of *ordered recombinants* containing `sorted_clusters2(jet_type)%n_ordered_recombinants` objects. This list for a given `jet_type` is only filled and accessible once the corresponding objects have been ordered by a previous cut. The order of an object corresponds to the position `i` in the list.

Accessing the member `ptr` of the recombinant and ordered-recombinant lists allows to perform a set of operations on particle momenta via some pre-defined methods, like computing the transverse momentum or the rapidity using `ptr%rapidity()` and `ptr%transverse_momentum()`, respectively. A more comprehensive list of available operations can be found by inspecting the file `/path/to/mocanlo/src/mocanlo/math/lorentz.f90`.

After this basic introduction, the example reported in Listing 32 can be understood as follows. The user-defined cut first checks if at least two tagged light jets (`jet_type==1`) exist. If not, the event is cut away. Otherwise, the function `mc_zep_cut_get_value` proceeds filling the list of `ordered_recombinants`, in case the jets were not already ordered by a previous cut. Finally, a Zeppenfeld variable among the positron, the leading, and the subleading jet is computed, and the event is cut away if this variable exceeds a hard-coded value.

Defining a new dynamical scale via the `user_defined_scales.F90` file requires to go through very similar steps as for the definition of a new cut. The user is expected to introduce a new scale type inheriting from `mc_scale` having its own `calc` procedure. As for the cuts, the latter has access to the particle momenta via the list of `sorted_clusters2`. One should keep in mind that when a scale is computed, the objects collected in `sorted_clusters2` are the ones that went through the complete sequence of cuts of `cut_card.xml`. It is the responsibility of the user to make sure that the objects used in the construction of the scale really exist and are well defined. The actual scale definition must be part of an appropriate `calc` procedure that returns the scale as a real variable. The user-defined scale can then be used either as an independent scale or as part of a `scale_sum` definition (see Sec. 7.4). It can be called from the input card like

```
<scales id="wzscatt">
    <scale_sum>
        ...
        <scale type="pt_user_scale">
            <name> my_scale  </name>
        </scale>
        ...
    </scale_sum>
    ...
</scales>
```

where `pt_user_scale` is the label used to initialise the scale template.

To introduce new histograms, the workflow is pretty similar up to some small differences that are illustrated in Listing 33.

```
module user_defined_histograms
  use observable_histogram
  implicit none

  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (1) Set number of user-defined histograms
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  integer :: n_user_histos = 2

  type, extends(mc_observable_histogram) :: mc_normalised_pt_difference_histogram
   contains
     procedure :: calc => mc_normalised_pt_difference_histogram_calc
  end type mc_normalised_pt_difference_histogram
  type, extends(mc_observable_histogram) :: mc_zep_histogram
   contains
     procedure :: calc => mc_zep_histogram_calc
  end type mc_zep_histogram
  ...

contains

  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (2) Add template to list
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  subroutine init_user_histos_templ(user_template,index)
    class(mc_observable_histogram), pointer :: user_template
    integer :: index

    if(index==1)then
      allocate (mc_normalised_pt_difference_histogram :: user_template)
      call user_template%init_template( &
          label="normalised_pt_difference", &
          latex_unit="", &
          x_min=0d0, &
          x_max=1d0, &
          bin_width=0.01d0, &
          plot_xmin=0d0, &
          plot_xmax=1d0, &
          logscale=.true. &
          )
    end if(index==2)then
      allocate (mc_zep_histogram :: user_template)
      ...
    end if
  end subroutine init_user_histos_templ

  ! !!!!!!!!!!!!!!!!!!!!!!!!!!
  ! (3) Define new histograms
  ! !!!!!!!!!!!!!!!!!!!!!!!!!!
  function mc_normalised_pt_difference_histogram_calc(this, partonic_process) &
   result(observable)
    class(mc_normalised_pt_difference_histogram) :: this
    type(mc_partonic_process), intent(in) :: partonic_process
    real(kind=dp) :: observable
    type(mc_recombinant) :: p_j1, p_j2
    real(kind=dp) :: pt_j1, pt_j2

    p_j1 = partonic_process%sorted_clusters2(1)%ordered_recombinants(1)%ptr
    p_j2 = partonic_process%sorted_clusters2(1)%ordered_recombinants(2)%ptr

    pt_j1 = p_j1%transverse_momentum()
    pt_j2 = p_j2%transverse_momentum()

    observable = ( pt_j1 - pt_j2 ) / ( pt_j1 + pt_j2 )
  end function mc_normalised_pt_difference_histogram_calc
```

```
    ...

end module user_defined_histograms
```

Listing 33: Example of a `user_defined_histograms.F90` file for the process $pp \to e^+e^-\mu^-\bar{\nu}_\mu jj$.

In the example above, two new histograms are introduced. The first thing to note is the definition of the template for the new types. In this case, the template contains a label together with an additional set of entries setting default values (that could always be overwritten from the input card, if needed) for the histogram ranges, binning and so on. The second difference is found in the `calc` procedure. There, the particle momenta must now be obtained through the `partonic_process` argument, that in turn provides access to the list of `sorted_clusters2`, already described above.

Once a histogram is defined, it can be used in the `plot_card.xml` as any other distribution:

```
<histograms id="wzscatt">
    ...
    <histogram type="normalised_pt_difference">
        <name>              "aptj"                          </name>
        <latex_title>       "Jet pt normalised difference" </latex_title>
        <latex_observable> "A_{p_{\text{T}, j}}"           </latex_observable>
    </histogram>
    ...
</histograms>
```

where in the example above the default values for all additional features of the histograms will be used as defined in the allocation of the template for the new type.

10. Conclusions

This manual describes the usage of MoCaNLO, a Monte Carlo integration code designed for arbitrary scattering processes at high-energy colliders. It allows the calculation of both NLO QCD and EW corrections as long as external particles that can give rise to real radiation are massless. Non-radiating final-state particles can be massive, and collinear singularities arising from initial-state electrons and muons can be regulated by their masses. For the calculation of tree-level and one-loop matrix elements, the program Recola is used, which in turn relies on Collier for the loop integrals. Infrared singularities are treated with the Catani–Seymour dipole formalism and its extensions, whereby all light fermions are assumed to be massless. Integrated and differential results are obtained upon combining different contributions like the Born, the virtual, the subtracted real and the integrated dipoles for a standard NLO calculation.

MoCaNLO has already been used for a large number of scattering processes at the LHC, including processes with up to eight particles in the final state. It can also deal with processes at lepton–lepton and lepton–proton colliders, as well as with ultra-peripheral collisions of photons in heavy-ion collisions. It supports the helicity selection for intermediate weak bosons and for incoming leptons.

In the future MoCaNLO could be extended in several respects. While MoCaNLO provides results for varied renormalisation and factorisation scales in an efficient way, it would be nice to also evaluate uncertainties from the parton-distributions functions efficiently. The treatment of processes with polarised resonances, which is currently subject to some limitations, is expected to be progressively extended and generalised. Presently, we are working on the matching of MoCaNLO predictions to parton showers via the MC@NLO method. Also, support for $1 \to n$ decay processes and $2 \to 1$ processes might be considered as an additional feature.

Acknowledgements

The first version of MoCaNLO was devised and implemented by Robert Feger. Notable contributions were made by Jean-Nicolas Lang, Stefan Rode, Timo Schmidt, and Christopher Schwan. We are indebted to Jean-Nicolas Lang and Sandro Uccirati for maintaining Recola. We thank Jose Luis Hernando Ariza and Jo Reimer for testing MoCaNLO.

This work is supported by the German Federal Ministry for Research, Technology and Space (BMFTR) under contracts no. 05H12WWE, 05H15WWCA1, 05H18WWCA1, 05H21WWCAA and the German Research Foundation (DFG) under reference numbers DE 623/6-1, DE 623/6-2, DE 623/8-1, and through the Research Training Group RTG2044. It was also supported by the state of Baden-Württemberg through bwHPC and the DFG through grant No. INST 39/963-1 FUGG (bwForCluster NEMO). This research has received funding from the European Research Council (ERC) under the EU Horizon 2020 Research and Innovation Programme (grant agreement no. 683211). The research of DL has also been partially supported by the Italian Ministry of Universities and Research (MUR) under the FIS grant (CUP: D53C24005480001, FLAME). GP is supported by the EU Horizon Europe Research and Innovation programme under the Marie-Skłodowska Curie Action (MSCA) "POEBLITA - POlarised Electroweak Bosons at the LHC with Improved Theoretical Accuracy", grant agreement no. 101149251 (CUP H45E2300129000).

A. Validated processes

This section contains a list of calculations that have been carried out with MoCaNLO, together with the respective references. The corresponding input cards are provided in the subfolders of the folder `validated_processes/`.

A.1. Top–antitop production and associated production

1. `pp_epvemumvmxbbx_qcd`
   The cards reproduce the calculation of NLO QCD corrections to off-shell tt production in Ref. [98].

2. `pp_epvemumvmxbbx_ew`
   The cards have been set up for the calculation of NLO EW corrections for off-shell tt production in Ref. [34].

3. `pp_epvemumvmxbbx_ew_DPA_ww`
   The cards have been set up for the calculation of NLO EW corrections for off-shell tt production in Ref. [34] in the WW DPA for the virtual part.

4. `pp_epvemumvmxbbx_ew_DPA_tt`
   The cards have been set up for the calculation of NLO EW corrections for off-shell tt production in Ref. [34] in the tt DPA for the virtual part.

5. `pp_epvemumvmxbbxh_qcd`
   The cards have been set up for the calculation of NLO QCD corrections for off-shell ttH production in Ref. [35]. Please also cite Ref. [33].

6. `pp_epvemumvmxbbxh_ew`
   The cards have been set up for the calculation of NLO EW corrections for off-shell ttH production in Ref. [35].

7. `pp_epvemumvmxbbxh_ew_DPA_ww`
   The cards have been set up for the calculation of NLO EW corrections for off-shell ttH production in Ref. [35] in the WW DPA for the virtual part.

8. `pp_epvemumvmxbbxh_ew_DPA_tt`
   The cards have been set up for the calculation of NLO EW corrections for off-shell ttH production in Ref. [35] in the tt DPA for the virtual part.

9. `pp_ttw`
   The cards have been set up for the calculation of the NLO corrections of orders $\mathcal{O}(\alpha_s^3\alpha^6)$, $\mathcal{O}(\alpha_s^2\alpha^7)$, and $\mathcal{O}(\alpha_s\alpha^8)$ to off-shell $t\bar{t}W^+$ production in the fully leptonic decay channel of Refs. [37, 39].

10. `pp_ttz`
    The cards have been set up for the calculation of the full set of LO and NLO corrections for off-shell $t\bar{t}Z$ production in Ref. [40].

11. `pp_ttbb`
    The cards have been set up for the calculation of the NLO QCD corrections for off-shell $t\bar{t}b\bar{b}$ production in Ref. [38].

A.2. Single-top production

1. `pp_tzj`
   The cards have been set up for the calculation of NLO QCD and EW corrections to associated tZj production in Ref. [58].

A.3. Vector-boson scattering

1. `pp_ssww_ew_ew`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to same-sign WW scattering in Ref. [50]. Please also cite Ref. [49] for the W^+W^+ signature.

2. `pp_ssww_ew_qcd`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to same-sign WW scattering in Ref. [50] for the W^+W^+ signature.

3. `pp_ssww_qcd_ew`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^5\alpha_s^2)$ corrections to same-sign WW scattering in Ref. [50] for the W^+W^+ signature.

4. `pp_ssww_qcd_qcd`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^4\alpha_s^3)$ corrections to same-sign WW scattering in Ref. [50] for the W^+W^+ signature.

5. `pp_ssww_ew_ew_minus_minus`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to same-sign WW scattering in Ref. [50] for the W^-W^- signature. Please also cite Refs. [49, 53].

6. `pp_ssww_ew_qcd_minus_minus`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to same-sign WW scattering in Ref. [50] for the W^-W^- signature.

7. `pp_ssww_ew_ew_pol`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to polarised W^+W^+ scattering in Ref. [65].

8. `pp_wz_ew_ew`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to W^+Z scattering in Ref. [51].

9. `pp_wz_ew_qcd`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to W^+Z scattering in Ref. [51].

10. `pp_wz_ew_ew_minus`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to W^-Z scattering in Ref. [51].

11. `pp_wz_ew_qcd_minus`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to W^-Z scattering in Ref. [51].

12. `pp_wz_ew_ew_pol`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to polarised W^+Z scattering in Ref. [69].

13. `pp_zz_ew_ew`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to ZZ scattering in Ref. [55]. Please also cite Ref. [54].

14. `pp_zz_ew_qcd`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to ZZ scattering in Ref. [55]. Please also cite Ref. [54].

15. `pp_zz_qcd_ew`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^5\alpha_s^2)$ corrections to ZZ scattering in Ref. [55].

16. `pp_zz_qcd_qcd`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^4\alpha_s^3)$ corrections to ZZ scattering in Ref. [55].

17. `pp_wpwm_ew_ew`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to W^+W^- scattering in Ref. [56].

18. `pp_wpwm_ew_qcd`
    The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to W^+W^- scattering in Ref. [56].

A.4. Triboson production

1. `pp_www_ew_ew`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^7)$ corrections to WWW production with one W boson decaying hadronically in Ref. [47]. Please also cite Ref. [49].

2. `pp_www_ew_qcd`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^6\alpha_s)$ corrections to WWW production with one W boson decaying hadronically in Ref. [47].

3. `pp_www_qcd_ew`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^5\alpha_s^2)$ corrections to WWW production with one W boson decaying hadronically in Ref. [47].

4. `pp_www_qcd_qcd`
   The cards have been set up for the calculation of $\mathcal{O}(\alpha^4\alpha_s^3)$ corrections to WWW production with one W boson decaying hadronically in Ref. [47].

5. `wzv_semi_leptonic`
   The folder collects the cards for the different contributions to WVZ production, with V being either a W or a Z boson decaying hadronically, with the setup described in Ref. [48]. The subfolder `pp_wvw_ew_ew` contains the $\mathcal{O}(\alpha^6)$ and the $\mathcal{O}(\alpha^7)$ corrections, while the subfolder `pp_wvw_ew_qcd` the $\mathcal{O}(\alpha^6\alpha_s)$ corrections. The LO contributions at $\mathcal{O}(\alpha^5\alpha_s)$ and $\mathcal{O}(\alpha^4\alpha_s^2)$ can be found in `pp_wvw_LO_as1_a5` and `pp_wvw_LO_as2_a4`, respectively. Cards to reproduce the results for the bottom- and photon-induced channels at the perturbative orders described in Ref. [48] are located in the subfolder `pp_wvw_bottoms` and `pp_wvw_photons`, respectively.

A.5. Diboson production

1. `pp_ww_pol`
   The cards have been set up for the calculation of NLO EW corrections to inclusive W^+W^- production at the LHC with polarised bosons in the fully leptonic channel [63]. Please also cite Ref. [59].

2. `pp_wz_pol`
   The cards have been set up for the calculation of NLO QCD and NLO EW corrections to inclusive W^+Z production at the LHC with polarised bosons in the fully leptonic channel [68]. Please also cite Ref. [60].

3. `pp_zz_pol`
   The cards have been set up for the calculation of NLO QCD and NLO EW corrections, as well as the loop-induced gg contribution, to inclusive ZZ production at the LHC with polarised bosons in the fully leptonic channel [66]. Please also cite Ref. [61].

A.6. Higgs production

1. `vbf_h`
   It contains the cards of Ref. [46] for Higgs production via vector-boson fusion (VBF) in one of the setups considered in the reference. It provides the NLO QCD and NLO EW corrections in the full computation to VBF in addition to the subleading contributions of orders $\mathcal{O}(\alpha_s^2\alpha^2)$, $\mathcal{O}(\alpha_s^3\alpha^2)$, and $\mathcal{O}(\alpha_s^4\alpha)$. Please also cite Ref. [45] when relevant, as indicated in the README files.

2. `vbf_hh`
   It contains cards for di-Higgs production via VBF from Ref. [45] at NLO QCD and NLO EW. In addition, cards for an inclusive set-up are also available for NLO EW corrections. In particular, these are used by the LHC Higgs Cross Section Working Group as canonical predictions for VBF.

A.7. Ultra-peripheral collisions

1. `tautau`
   The cards have been set up for one check of the calculation of EW corrections to tau-pair production in UPC [99], namely on-shell massless tau fermions leptons.

References

[1] J. Alwall, et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, *JHEP* **07** (2014) 079, [arXiv:1405.0301].

[2] S. Alioli, P. Nason, C. Oleari, and E. Re, *A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX*, *JHEP* **06** (2010) 043, [arXiv:1002.2581].

[3] **Sherpa** Collaboration, E. Bothmann et al., *Event generation with Sherpa 3*, *JHEP* **12** (2024) 156, [arXiv:2410.22148].

[4] G. Bewick et al., *Herwig 7.3 release note*, *Eur. Phys. J. C* **84** (2024) 1053, [arXiv:2312.05175].

[5] 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].

[6] F. A. Berends, R. Pittau, and R. Kleiss, *All electroweak four fermion processes in electron - positron collisions*, *Nucl. Phys. B* **424** (1994) 308–342, [hep-ph/9404313].

[7] A. Denner, S. Dittmaier, M. Roth, and D. Wackeroth, *Predictions for all processes* $e^+e^- \to 4\,\text{fermions} + \gamma$, *Nucl. Phys.* **B560** (1999) 33–65, [hep-ph/9904472].

[8] M. Roth, *Precise predictions for four fermion production in electron positron annihilation*, PhD thesis, ETH Zürich, 1999.

[9] S. Dittmaier and M. Roth, *LUSIFER: A LUcid approach to six FERmion production*, *Nucl. Phys.* **B642** (2002) 307–343, [hep-ph/0206070].

[10] S. Actis, A. Denner, L. Hofer, A. Scharf, and S. Uccirati, *Recursive generation of one-loop amplitudes in the Standard Model*, *JHEP* **04** (2013) 037, [arXiv:1211.6316].

[11] S. Actis, et al., *RECOLA: REcursive Computation of One-Loop Amplitudes*, *Comput. Phys. Commun.* **214** (2017) 140–173, [arXiv:1605.01090].

[12] A. Denner, S. Dittmaier, and L. Hofer, *COLLIER: a fortran-based Complex One-Loop LIbrary in Extended Regularizations*, *Comput. Phys. Commun.* **212** (2017) 220–238, [arXiv:1604.06792].

[13] S. Catani and M. Seymour, *A general algorithm for calculating jet cross-sections in NLO QCD*, *Nucl. Phys. B* **485** (1997) 291–419, [hep-ph/9605323]. [Erratum: *Nucl. Phys. B* **510** (1998) 503–504].

[14] Z. Nagy and Z. Trócsányi, *Next-to-leading order calculation of four jet observables in electron positron annihilation*, *Phys. Rev. D* **59** (1999) 014020, [hep-ph/9806317]. [Erratum: Phys.Rev.D 62, 099902 (2000)].

[15] S. Dittmaier, *A general approach to photon radiation off fermions*, *Nucl. Phys. B* **565** (2000) 69–122, [hep-ph/9904440].

[16] S. Catani, S. Dittmaier, M. H. Seymour, and Z. Trócsányi, *The dipole formalism for next-to-leading order QCD calculations with massive partons*, *Nucl. Phys. B* **627** (2002) 189–265, [hep-ph/0201036].

[17] S. Dittmaier, A. Kabelschacht, and T. Kasprzik, *Polarized QED splittings of massive fermions and dipole subtraction for non-collinear-safe observables*, *Nucl. Phys.* **B800** (2008) 146–189, [arXiv:0802.1405].

[18] L. Basso, S. Dittmaier, A. Huss, and L. Oggero, *Techniques for the treatment of IR divergences in decay processes at NLO and application to the top-quark decay*, *Eur. Phys. J.* **C76** (2016) 56, [arXiv:1507.04676].

[19] S. Frixione, *Isolated photons in perturbative QCD*, *Phys. Lett. B* **429** (1998) 369–374, [hep-ph/9801442].

[20] E. W. N. Glover and A. G. Morgan, *Measuring the photon fragmentation function at LEP*, *Z. Phys. C* **62** (1994) 311–322.

[21] A. Denner, S. Dittmaier, T. Gehrmann, and C. Kurz, *Electroweak corrections to hadronic event shapes and jet production in e^+e^- annihilation*, *Nucl. Phys.* **B836** (2010) 37–90, [arXiv:1003.0986].

[22] A. Denner, S. Dittmaier, M. Hecht, and C. Pasold, *NLO QCD and electroweak corrections to $W + \gamma$ production with leptonic W-boson decays*, *JHEP* **04** (2015) 018, [arXiv:1412.7421].

[23] A. Denner, S. Dittmaier, M. Pellen, and C. Schwan, *Low-virtuality photon transitions $\gamma^* \to f\bar{f}$ and the photon-to-jet conversion function*, *Phys. Lett. B* **798** (2019) 134951, [arXiv:1907.02366].

[24] A. Denner, S. Dittmaier, M. Roth, and L. H. Wieders, *Electroweak corrections to charged-current $e^+e^- \to 4$ fermion processes: Technical details and further results*, *Nucl. Phys. B* **724** (2005) 247–294, [hep-ph/0505042]. [Erratum: *Nucl.Phys. B* **854**, 504–507 (2012)].

[25] A. Denner and S. Dittmaier, *The complex-mass scheme for perturbative calculations with unstable particles*, *Nucl. Phys. B Proc. Suppl.* **160** (2006) 22–26, [hep-ph/0605312].

[26] A. Denner and S. Dittmaier, *Electroweak Radiative Corrections for Collider Physics*, *Phys. Rept.* **864** (2020) 1–163, [arXiv:1912.06823].

[27] R. G. Stuart, *Gauge invariance, analyticity and physical observables at the Z^0 resonance*, *Phys. Lett. B* **262** (1991) 113–119.

[28] A. Aeppli, G. J. van Oldenborgh, and D. Wyler, *Unstable particles in one loop calculations*, *Nucl. Phys. B* **428** (1994) 126–146, [hep-ph/9312212].

[29] A. Denner, S. Dittmaier, M. Roth, and D. Wackeroth, *Electroweak radiative corrections to $e^+e^- \to WW \to 4$ fermions in double pole approximation: The RACOONWW approach*, *Nucl. Phys. B* **587** (2000) 67–117, [hep-ph/0006307].

[30] A. Denner, S. Dittmaier, and M. Roth, *Non-factorizable photonic corrections to $e^+e^- \to WW \to$ four fermions*, *Nucl. Phys. B* **519** (1998) 39–84, [hep-ph/9710521].

[31] E. Accomando, A. Denner, and A. Kaiser, *Logarithmic electroweak corrections to gauge-boson pair production at the LHC*, *Nucl. Phys. B* **706** (2005) 325–371, [hep-ph/0409247].

[32] S. Dittmaier and C. Schwan, *Non-factorizable photonic corrections to resonant production and decay of many unstable particles*, *Eur. Phys. J. C* **76** (2016) 144, [arXiv:1511.01698].

[33] A. Denner and R. Feger, *NLO QCD corrections to off-shell top-antitop production with leptonic decays in association with a Higgs boson at the LHC*, *JHEP* **11** (2015) 209, [arXiv:1506.07448].

[34] A. Denner and M. Pellen, *NLO electroweak corrections to off-shell top-antitop production with leptonic decays at the LHC*, *JHEP* **08** (2016) 155, [arXiv:1607.05571].

[35] A. Denner, J.-N. Lang, M. Pellen, and S. Uccirati, *Higgs production in association with off-shell top-antitop pairs at NLO EW and QCD at the LHC*, *JHEP* **02** (2017) 053, [arXiv:1612.07138].

[36] A. Denner and M. Pellen, *Off-shell production of top-antitop pairs in the lepton+jets channel at NLO QCD*, *JHEP* **02** (2018) 013, [arXiv:1711.10359].

[37] A. Denner and G. Pelliccioli, *NLO QCD corrections to off-shell $\bar{t}tW^+$ production at the LHC*, *JHEP* **11** (2020) 069, [arXiv:2007.12089].

[38] A. Denner, J.-N. Lang, and M. Pellen, *Full NLO QCD corrections to off-shell ttbb production*, *Phys. Rev. D* **104** (2021) 056018, [arXiv:2008.00918].

[39] A. Denner and G. Pelliccioli, *Combined NLO EW and QCD corrections to off-shell $\bar{t}tW$ production at the LHC*, *Eur. Phys. J. C* **81** (2021) 354, [arXiv:2102.03246].

[40] A. Denner, D. Lombardi, and G. Pelliccioli, *Complete NLO corrections to off-shell $\bar{t}tZ$ production at the LHC*, *JHEP* **09** (2023) 072, [arXiv:2306.13535].

[41] A. Denner, M. Pellen, and G. Pelliccioli, *NLO QCD corrections to off-shell top–antitop production with semi-leptonic decays at lepton colliders*, *Eur. Phys. J. C* **83** (2023) 353, [arXiv:2302.04188].

[42] M. Czakon, A. Mitov, M. Pellen, and R. Poncelet, *A detailed investigation of W+c-jet at the LHC*, *JHEP* **02** (2023) 241, [arXiv:2212.00467].

[43] B. Biedermann, et al., *Automation of NLO QCD and EW corrections with Sherpa and Recola*, *Eur. Phys. J.* **C77** (2017) 492, [arXiv:1704.05783].

[44] S. Bräuer, A. Denner, M. Pellen, M. Schönherr, and S. Schumann, *Fixed-order and merged parton-shower predictions for WW and WWj production at the LHC including NLO QCD and EW corrections*, *JHEP* **10** (2020) 159, [arXiv:2005.12128].

[45] F. A. Dreyer, A. Karlberg, J.-N. Lang, and M. Pellen, *Precise predictions for double-Higgs production via vector-boson fusion*, *Eur. Phys. J. C* **80** (2020) 1037, [arXiv:2005.13341].

[46] G. Barone et al., *Higgs production via vector-boson fusion at the LHC*, arXiv:2507.22574.

[47] A. Denner, M. Pellen, M. Schönherr, and S. Schumann, *Tri-boson and WH production in the W^+W^+jj channel: predictions at full NLO accuracy and beyond*, *JHEP* **08** (2024) 043, [arXiv:2406.11516].

[48] A. Denner, D. Lombardi, S. L. P. Chavez, and G. Pelliccioli, *NLO corrections to triple vector-boson production in final states with three charged leptons and two jets*, *JHEP* **09** (2024) 187, [arXiv:2407.21558].

[49] B. Biedermann, A. Denner, and M. Pellen, *Large electroweak corrections to vector-boson scattering at the Large Hadron Collider*, *Phys. Rev. Lett.* **118** (2017) 261801, [arXiv:1611.02951].

[50] B. Biedermann, A. Denner, and M. Pellen, *Complete NLO corrections to W^+W^+ scattering and its irreducible background at the LHC*, *JHEP* **10** (2017) 124, [arXiv:1708.00268].

[51] A. Denner, S. Dittmaier, P. Maierhöfer, M. Pellen, and C. Schwan, *QCD and electroweak corrections to WZ scattering at the LHC*, *JHEP* **06** (2019) 067, [arXiv:1904.00882].

[52] M. Pellen, *Exploring the scattering of vector bosons at LHCb*, *Phys. Rev. D* **101** (2020) 013002, [arXiv:1908.06805].

[53] M. Chiesa, A. Denner, J.-N. Lang, and M. Pellen, *An event generator for same-sign W-boson scattering at the LHC including electroweak corrections*, *Eur. Phys. J. C* **79** (2019) 788, [arXiv:1906.01863].

[54] A. Denner, R. Franken, M. Pellen, and T. Schmidt, *NLO QCD and EW corrections to vector-boson scattering into ZZ at the LHC*, *JHEP* **11** (2020) 110, [arXiv:2009.00411].

[55] A. Denner, R. Franken, M. Pellen, and T. Schmidt, *Full NLO predictions for vector-boson scattering into Z bosons and its irreducible background at the LHC*, *JHEP* **10** (2021) 228, [arXiv:2107.10688].

[56] A. Denner, R. Franken, T. Schmidt, and C. Schwan, *NLO QCD and EW corrections to vector-boson scattering into W^+W^- at the LHC*, *JHEP* **06** (2022) 098, [arXiv:2202.10844].

[57] A. Denner, D. Lombardi, and C. Schwan, *Double-pole approximation for leading-order semi-leptonic vector-boson scattering at the LHC*, *JHEP* **08** (2024) 146, [arXiv:2406.12301].

[58] A. Denner, G. Pelliccioli, and C. Schwan, *NLO QCD and EW corrections to off-shell tZj production at the LHC*, *JHEP* **10** (2022) 125, [arXiv:2207.11264].

[59] A. Denner and G. Pelliccioli, *Polarized electroweak bosons in W^+W^- production at the LHC including NLO QCD effects*, *JHEP* **09** (2020) 164, [arXiv:2006.14867].

[60] A. Denner and G. Pelliccioli, *NLO QCD predictions for doubly-polarized WZ production at the LHC*, *Phys. Lett. B* **814** (2021) 136107, [arXiv:2010.07149].

[61] A. Denner and G. Pelliccioli, *NLO EW and QCD corrections to polarized ZZ production in the four-charged-lepton channel at the LHC*, *JHEP* **10** (2021) 097, [arXiv:2107.06579].

[62] A. Denner, C. Haitz, and G. Pelliccioli, *NLO QCD corrections to polarized diboson production in semileptonic final states*, *Phys. Rev. D* **107** (2023) 053004, [arXiv:2211.09040].

[63] A. Denner, C. Haitz, and G. Pelliccioli, *NLO EW corrections to polarised W^+W^- production and decay at the LHC*, *Phys. Lett. B* **850** (2024) 138539, [arXiv:2311.16031].

[64] M. Grossi, G. Pelliccioli, and A. Vicini, *From angular coefficients to quantum observables: a phenomenological appraisal in di-boson systems*, *JHEP* **12** (2024) 120, [arXiv:2409.16731].

[65] A. Denner, C. Haitz, and G. Pelliccioli, *NLO EW and QCD corrections to polarised same-sign WW scattering at the LHC*, *JHEP* **11** (2024) 115, [arXiv:2409.03620].

[66] C. Carrivale et al., *Precise Standard-Model predictions for polarised Z-boson pair production and decay at the LHC*, *Eur. Phys. J. C* **85** (2025) 1342, [arXiv:2505.09686].

[67] M. Del Gratta, et al., *Z-boson quantum tomography at next-to-leading order*, *JHEP* **02** (2026) 056, [arXiv:2509.20456].

[68] G. Pelliccioli and R. Poncelet, *Precise predictions for joint polarisation fractions in WZ production at the LHC*, arXiv:2510.25898.

[69] A. Denner, R. Franken, C. Haitz, D. Lombardi, and G. Pelliccioli, *Electroweak corrections to doubly polarised WZ scattering at the LHC*, arXiv:2510.26462.

[70] A. Buckley, et al., *LHAPDF6: parton density access in the LHC precision era*, *Eur. Phys. J. C* **75** (2015) 132, [arXiv:1412.7420].

[71] H.-S. Shao and D. d'Enterria, *gamma-UPC: automated generation of exclusive photon-photon processes in ultraperipheral proton and nuclear collisions with varying form factors*, *JHEP* **09** (2022) 248, [arXiv:2207.03012].

[72] J. A. M. Vermaseren, *Axodraw*, *Comput. Phys. Commun.* **83** (1994) 45–58.

[73] D. Binosi and L. Theußl, *JaxoDraw: A Graphical user interface for drawing Feynman diagrams*, *Comput. Phys. Commun.* **161** (2004) 76–86, [hep-ph/0309015].

[74] K. Melnikov and O. I. Yakovlev, *Top near threshold: all α_S corrections are trivial*, *Phys. Lett.* **B324** (1994) 217–223, [hep-ph/9302311].

[75] V. S. Fadin, V. A. Khoze, and A. D. Martin, *Interference radiative phenomena in the production of heavy unstable particles*, *Phys. Rev.* **D49** (1994) 2247–2256.

[76] V. S. Fadin, V. A. Khoze, and A. D. Martin, *How suppressed are the radiative interference effects in heavy instable particle production?*, *Phys. Lett.* **B320** (1994) 141–144, [hep-ph/9309234].

[77] A. Denner, S. Dittmaier, and M. Roth, *Further numerical results on nonfactorizable corrections to $e^+e^- \to W^+W^- \to$ four fermions*, *Phys. Lett. B* **429** (1998) 145–150, [hep-ph/9803306].

[78] D. Yu. Bardin, A. Leike, T. Riemann, and M. Sachwitz, *Energy-dependent width effects in e^+e^- annihilation near the Z-boson pole*, *Phys. Lett.* **B206** (1988) 539–542.

[79] V. Bertone, et al., *Improving methods and predictions at high-energy e^+e^- colliders within collinear factorisation*, *JHEP* **10** (2022) 089, [arXiv:2207.03265].

[80] W. Beenakker et al., *WW cross-sections and distributions*, in *Physics at LEP2* (G. Altarelli, T. Sjöstrand, F. Zwirner, ed.), vol. 1, (Geneva), pp. 79–139, CERN, 1996. hep-ph/9602351. CERN-96-01.

[81] Z. Nagy, *Next-to-leading order calculation of three jet observables in hadron hadron collision*, *Phys. Rev. D* **68** (2003) 094002, [hep-ph/0307268].

[82] F. James, *RANLUX: A FORTRAN implementation of the high quality pseudorandom number generator of Luscher*, *Comput. Phys. Commun.* **79** (1994) 111–114. [Erratum: *Comput.Phys.Commun. 97* **357** (1996)].

[83] R. N. Cahn and J. D. Jackson, *Realistic equivalent photon yields in heavy ion collisions*, *Phys. Rev. D* **42** (1990) 3690–3695.

[84] M. Vidovic, M. Greiner, C. Best, and G. Soff, *Impact parameter dependence of the electromagnetic particle production in ultrarelativistic heavy ion collisions*, *Phys. Rev. C* **47** (1993) 2308–2319.

[85] R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, *Comput. Phys. Commun.* **83** (1994) 141–146, [hep-ph/9405257].

[86] M. Cacciari, G. P. Salam, and G. Soyez, *The anti-k_t jet clustering algorithm*, *JHEP* **04** (2008) 063, [arXiv:0802.1189].

[87] Y. L. Dokshitzer, G. D. Leder, S. Moretti, and B. R. Webber, *Better jet clustering algorithms*, *JHEP* **08** (1997) 001, [hep-ph/9707323].

[88] M. Wobisch and T. Wengler, *Hadronization corrections to jet cross-sections in deep inelastic scattering*, in *Workshop on Monte Carlo Generators for HERA Physics (Plenary Starting Meeting)*, pp. 270–279, 4, 1998. hep-ph/9907280.

[89] S. Catani, Y. L. Dokshitzer, M. H. Seymour, and B. R. Webber, *Longitudinally invariant $k_\perp$-clustering algorithms for hadron hadron collisions*, *Nucl. Phys. B* **406** (1993) 187–224.

[90] S. D. Ellis and D. E. Soper, *Successive combination jet algorithm for hadron collisions*, *Phys. Rev. D* **48** (1993) 3160–3166, [hep-ph/9305266].

[91] E. W. N. Glover and A. G. Morgan, *Soft gluon radiation in photon plus single jet events at LEP*, *Phys. Lett. B* **324** (1994) 487–491.

[92] **ALEPH** Collaboration, D. Buskulic et al., *First measurement of the quark to photon fragmentation function*, *Z. Phys.* **C69** (1996) 365–378.

[93] A. Denner, S. Dittmaier, T. Kasprzik, and A. Mück, *Electroweak corrections to W+jet hadroproduction including leptonic W-boson decays*, *JHEP* **08** (2009) 075, [arXiv:0906.1656].

[94] A. Denner, S. Dittmaier, T. Kasprzik, and A. Mück, *Electroweak corrections to dilepton + jet production at hadron colliders*, *JHEP* **06** (2011) 069, [arXiv:1103.0914].

[95] A. Denner, L. Hofer, A. Scharf, and S. Uccirati, *Electroweak corrections to lepton pair production in association with two hard jets at the LHC*, *JHEP* **01** (2015) 094, [arXiv:1411.0916].

[96] A. Denner, S. Dittmaier, M. Hecht, and C. Pasold, *NLO QCD and electroweak corrections to $Z + \gamma$ production with leptonic Z-boson decays*, *JHEP* **02** (2016) 057, [arXiv:1510.08742].

[97] A. Keshavarzi, D. Nomura, and T. Teubner, *Muon $g-2$ and $\alpha(M_Z^2)$: a new data-based analysis*, *Phys. Rev. D* **97** (2018) 114025, [arXiv:1802.02995].

[98] A. Denner, S. Dittmaier, S. Kallweit, and S. Pozzorini, *NLO QCD corrections to off-shell top-antitop production with leptonic decays at hadron colliders*, *JHEP* **10** (2012) 110, [arXiv:1207.5018].

[99] S. Dittmaier, T. Engel, J. L. H. Ariza, and M. Pellen, *Electroweak corrections to $\tau^+\tau^-$ production in ultraperipheral heavy-ion collisions at the LHC*, *JHEP* **08** (2025) 051, [arXiv:2504.11391].