Eur. Phys. J. C (2026) 86:22
https://doi.org/10.1140/epjc/s10052-025-15185-8

THE EUROPEAN PHYSICAL JOURNAL C

Regular Article - Theoretical Physics

Colibri
A new tool for fast-flying PDF fits

Mark N. Costantini[1,a], **Luca Mantani**[3,b], **James M. Moore**[2,c], **Valentina Schütze Sánchez**[1,d], **Maria Ubiali**[1,e]

[1] DAMTP, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK
[2] Lucy Cavendish College, Lady Margaret Road, Cambridge CB3 0BU, UK
[3] Instituto de Fisica Corpuscular (IFIC), Universidad de Valencia-CSIC, 46980 Valencia, Spain

Received: 20 October 2025 / Accepted: 5 December 2025
© The Author(s) 2026

Abstract We present Colibri, an open-source Python code that provides a general and flexible tool for PDF fits. The code is built so that users can implement their own PDF model, and use the built-in functionalities of Colibri for a fast computation of observables. It grants easy access to experimental data, several error propagation methodologies, including the Hessian method, the Monte Carlo replica method, and an efficient numerical Bayesian sampling algorithm. To demonstrate the capabilities of Colibri, we consider its simplest application: a polynomial PDF parametrisation. We perform closure tests using a full set of DIS data and compare the results of Hessian and Monte Carlo fits with those from a Bayesian fit. We further discuss how the functionalities illustrated in this example can be extended to more complex PDF parametrisations. In particular, the Bayesian framework in Colibri provides a principled approach to model selection and model averaging, making it a valuable tool for benchmarking and combining different PDF parametrisations on solid statistical grounds.

Contents

1 Introduction
2 Colibri: a PDF building platform
3 Case study: a simple Colibri fit with Les Houches PDFs
4 Conclusions
Appendix A: Bayesian update
Appendix B: Bayesian linear regression
Appendix C: How to implement a PDF model in Colibri
Appendix D: The Les Houches parametrisation
References

1 Introduction

A precise determination of the subnuclear structure of the proton is a central challenge in high-energy physics. Parton Distribution Functions (PDFs), which, to a first approximation, encode the momentum distributions of quarks, anti-

[a] e-mail: mnc33@cam.ac.uk
[b] e-mail: luca.mantani@uv.es
[c] e-mail: jmm232@cam.ac.uk
[d] e-mail: vcs32@cam.ac.uk (corresponding author)
[e] e-mail: M.Ubiali@damtp.cam.ac.uk

quarks and gluons inside the proton, are key ingredients in precision predictions for hadron collider experiments such as the Large Hadron Collider (LHC). Extracting PDFs from data is, however, a highly non-trivial statistical inverse problem that must reconcile theoretical constraints with a wealth of experimental measurements, together with their uncertainties and correlations [1–4].

Over the years, different inference strategies have been developed to address this task. The Hessian method, used in the first PDF fits producing error sets, provides an efficient framework for uncertainty estimate by approximating the likelihood surface as a quadratic function of parameters around the minimum, so that error propagation reduces to linear operations involving the covariance matrix of the fit parameters. This approach is computationally inexpensive for a moderately large parameter space and it is well suited for problems where the likelihood is close to Gaussian. However, it may produce unfaithful uncertainties when non-Gaussianities or parameter degeneracies are present. Several PDF fitting collaborations adopt the Hessian approach for error propagation [5–8], typically alongside a polynomial parametrisation of PDFs, and some collaborations introduce the concept of *tolerance*, which amounts to obtaining the uncertainty on the parameters by setting $\Delta\chi^2 = T^2$ instead of the standard $\Delta\chi^2 = 1$ in order to account for inconsistencies in a global PDF fit,[1] [10–13].

Monte Carlo replica methods, on the other hand, can be applied to an arbitrarily large parameter space, provided a cross-validation mechanism is in place. It relies on the generation of a large ensemble of *pseudodata replicas* that accurately reproduce the full experimental uncertainties and the correlations of the data used to determine PDFs.[2] Each pseudo-data replica is fitted independently, and the statistical spread of the resulting PDF ensemble provides an estimate of the PDF uncertainties and of non-Gaussianities in the PDF ensemble. It was recently highlighted [20] that the Monte Carlo method faithfully estimates uncertainties in the linear regime, but it might be unreliable in the presence of non-linearities. The NNPDF [14,21,22] and JAM [23] collaborations both use the Monte Carlo method to determine PDF uncertainties, the former alongside a redundant neural network parametrisation, the latter using a polynomial parametrisation for PDFs.

The Hessian and Monte Carlo approaches are both widely used in global PDF analyses, and there are well established ways to convert a Hessian PDF set into a Monte Carlo one and vice-versa [24–29], that have been used to produce the two most recent PDF4LHC combinations [30,31] and the MSHT and NNPDF combination of the approximate N3LO sets [32]. By contrast, Bayesian inference, despite its ability to incorporate prior knowledge, quantify uncertainties probabilistically, and naturally accommodate theoretical constraints, has so far played only a limited role in practical PDF fitting. While some works have paved the way forward [33–38], a full-fledged PDF fit is yet to be produced, in part due to the absence of accessible, general-purpose tools.

In this work we introduce `Colibri`, a new open-source platform written in Python, designed to perform global PDF fits with a unified treatment of inference. A key feature of our framework is its flexibility: `Colibri` can accommodate any PDF parametrisation defined by the user, and supports multiple inference methods, namely Hessian, Monte Carlo, and Bayesian fits, within a common infrastructure. This makes it possible to easily benchmark the same parametrisation across different methodologies, a capability that is unique to our approach and enhances both flexibility and robustness. Particularly novel is the integration of Bayesian inference, implemented through modern nested-sampling algorithms, which allow for the efficient exploration of high-dimensional parameter spaces and a probabilistic characterisation of uncertainties.

While existing tools such as `xFitter` [39,40] have played an important role in making PDF determinations more accessible, their scope is more restricted in key respects, as they only provide a limited set of parametrisations, and focus on two error-propagation strategies. The public release of the `NNPDF` code [41], on the other hand, has made the full-fledged NNPDF methodology, analysis tools, theory predictions and experimental data available to all users, making results reproducible and replicable. However, it is limited to the specific parametrisation and error propagation adopted by the NNPDF collaboration since the release of the `NNPDF4.0` global fit [21].

In contrast, `Colibri`, builds on the availability of data and a fast interface with theory predictions from the `NNPDF` public code [41,42] and it extends the current paradigm by providing a fully modular infrastructure, allowing users to freely choose both the parametrisation and the inference strategy. This comprehensive approach not only broadens the range of possible applications, but also enables direct and systematic comparisons between methodologies, a feature that is essential for advancing the robustness and reproducibility of global PDF analyses. Moreover, the Bayesian framework in `Colibri` enables the statistical combination of different PDF models based on their Bayesian evidence [33]. This feature makes `Colibri` a valuable tool for benchmarking and combining PDF sets derived from diverse fitting methodologies on a rigorous statistical footing.

The paper is structured as follows. Section 2 describes the architecture and implementation of `Colibri`, includ-

[1] In the MSHT approach this is refined using a *dynamical tolerance* which is selected via a hyperoptimisation procedure [9].

[2] In recent NNPDF studies the Monte Carlo replica sample incorporates theory uncertainties due to missing higher order uncertainties (MHOUs) and nuclear uncertainties into their PDF error propagation [14–19].

ing the statistical algorithms and computational optimisations employed. In Sect. 3, we validate our framework with benchmark fits to existing datasets and compare the outcomes across inference methods. Finally, Sect. 4 summarises our findings and outlines future directions for development and applications.

2 `Colibri`: a PDF building platform

In this section we describe the architecture and core functionalities of `Colibri`, highlighting how this Python framework allows users to implement arbitrary PDF parametrisations and fit them with a range of inference strategies.

The design philosophy behind `Colibri` is built on three pillars: (i) *modularity*, as the tasks of defining a PDF model, constructing the likelihood, interfacing with data and theory predictions, and performing inference are separated into clear components with minimal assumptions about their specific form; (ii) *performance*, as the code is made to be fast and efficient by leveraging JAX's [43] high-performance array operations and native GPU support for fast computation; (iii) *universality*, as all models share the same inference methods as well as data and theory predictions. Modularity enables the user to benchmark different parametrisations under identical statistical conditions, and to test the impact of alternative inference strategies on the same model. High-level performance and universality enable users to perform reliable comparisons and rigorous methodological studies. Figure 1 shows a schematic overview of the code's workflow, showcasing the role that each of these modular building blocks play in its general functioning.

The section is organised as follows. We begin by introducing the `PDFModel` base class, which provides a uniform interface for all PDF parametrisations (Sect. 2.1). We then discuss the likelihood function and the implementation of theoretical and experimental constraints (Sect. 2.2). Next, we describe how data and theory predictions are incorporated, and how the forward map from PDF parameters to observables is built. Finally, we review the inference methods available in `Colibri`, including Hessian, Monte Carlo, and Bayesian error propagation.

2.1 The `Colibri` PDF model class

To separate the definition of PDF parametrisations from their numerical inference, `Colibri` provides the abstract base class `PDFModel`. Listing 1 shows pseudo-code illustrating its structure. At a minimum, a `PDFModel` in `Colibri` must specify:

- a list of model parameters, representing the degrees of freedom of the PDF;
- a method to map the model parameters to the values of the PDFs on a specified grid in momentum fraction x for each parton flavour.

Such an abstraction allows users to implement a wide range of model architectures, from simple parametric forms to neural network based approaches, while leaving performance-critical tasks such as convolutions with pre-tabulated kernels and parameter sampling to optimised external engines.

This means that, in practice, the user can implement a new parametrisation within `Colibri` by completing the abstract `PDFModel` class and specifying the two required methods; parameter specification and forward map. A detailed example of this procedure is given in the step-by-step tutorial outlined in Appendix C. We will now give an overview of what these methods are.

```
class PDFModel(ABC):
    """An abstract class describing the
    key features of a PDF model."""

    name = "Abstract PDFModel"

    @property
    @abstractmethod
    def param_names(self) -> list:
        pass

    @abstractmethod
    def grid_values_func(self, xgrid)
   -> Callable:
        """Produce a function that maps
    model parameters
        to PDF values on the grid '
   xgrid'.
        """
        pass

    def pred_and_pdf_func(
        self,
        xgrid,
        forward_map,
    ) -> Callable, Tuple:
        pdf_func = self.
   grid_values_func(xgrid)

        def pred_and_pdf(params,
   fast_kernel_arrays):
            pdf = pdf_func(params)
            predictions = forward_map(
   pdf, fast_kernel_arrays)
            return predictions, pdf

        return pred_and_pdf
```

Code Listing 1 Abstract PDFModel interface.

Parameter specification
Every PDF model must declare the parameters to be fitted (e.g. normalisations, small-/large-x exponents, polynomial coefficients, weights and biases of a neural network, weights of a linear combination, …). These are listed in the

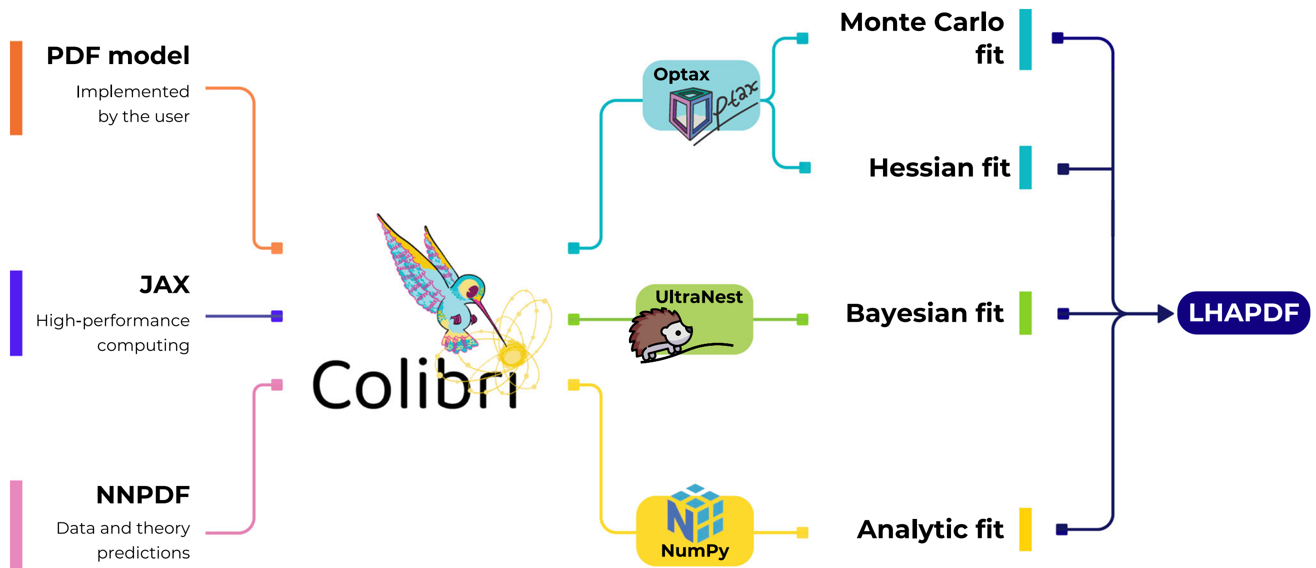


Fig. 1 `Colibri`'s workflow: the code takes as input **(i)** a PDF model, which may be any arbitrary parametrisation implemented by the user, makes use of **(ii)** JAX [43], which provides high-performance array operations and native GPU support for fast computations, and inherits **(iii)** data and theory predictions from the `NNPDF` public code [41,42]. The code then performs a fit using a given inference method, which is specified by the user. At the time of release, the options are a Monte Carlo, Hessian, Bayesian or analytic fit. In each case, the result is outputted in the LHAPDF format [44,45]

`param_names` property, which returns an ordered list of strings defining the parameter names in a fixed sequence.

Grid Evaluation Method
The core of the `PDFModel` class is the `grid_values_func` method, which returns a JAX-compatible function [43],

$$\mathbf{f}: \mathbb{R}^{N_\theta} \to \mathbb{R}^{N_{\text{fl}} \times N_{\text{x}}}, \quad \mathbf{f}: \boldsymbol{\theta} \mapsto \mathbf{f}(\boldsymbol{\theta}) \tag{2.1}$$

mapping an N_θ-dimensional parameter vector $\boldsymbol{\theta}$ into the PDF values[3] for each parton flavour index[4] evaluated on the user-provided x-grid of length N_x. In practice, for a standard PDF fit, the user only needs to define this method. The framework then automatically handles the construction of all the resources, such as the forward map from the parameters to the physical observables that enter the regression problem, i.e. the theory predictions, needed in a PDF fit. Note that at the moment sum rules need to be imposed at the level of the model implementation, since `Colibri` does not check or impose the sum rules via penalty terms. This means that, if the user wishes to enforce sum rules, they must incorporate them directly into their `model.py` definition, ensuring that the PDF, $\mathbf{f}(\boldsymbol{\theta})$, automatically satisfies them. A less strict alternative is to override the likelihood and introduce a penalty term that suppresses regions of parameter space violating the sum rules.

[3] Note that the f_{grid} method should actually return xPDF (that is, the PDF multiplied by the momentum fraction x).

[4] These are γ, Σ, g, V, V_3, V_8, V_{15}, V_{24}, V_{35}, T_3, T_8, T_{15}, T_{24} and T_{35}, since `Colibri` works in the evolution basis.

Forward map and theory predictions
To compute physical observables (structure functions, cross sections, etc.), one must convolve the PDFs with partonic cross sections computed at a given perturbative order in the QCD and EW expansions. In `Colibri` this is handled via the `pred_and_pdf_func` method, which takes again the N_{x}-dimensional x-grid and a forward map that projects the PDF parameters to the space of physical observables. The method boils down to a function taking as input the PDF parameters and a tuple of fast-kernel (FK) matrices, and outputs the theoretical predictions, i.e. an explicit function $\mathbf{f}$ of the parameters $\boldsymbol{\theta}$:

$$(\boldsymbol{\theta}, \text{FK}) \mapsto \big(\text{predictions}, \mathbf{f}(\boldsymbol{\theta})\big). \tag{2.2}$$

The matrix (FK) is called an FK-table in the NNPDF jargon, and provides a fast interpolation of the forward map, namely $(\text{FK})_{Ii} = \int c_I(x)\, p_i(x)\, dx$, where $c_I(x)$ are known functions computed in perturbation theory by convolving the PDF DGLAP evolution kernels and the partonic cross sections and $p_i(x)$ is an interpolation polynomial, relative to the point x_i. Our notation reflects this convention. The FK arrays (i) evaluate the PDF on the grid via `grid_values_func`, and (ii) feed the resulting $N_{\text{fl}} \times N_{\text{x}}$ array into the supplied `forward_map` to yield a 1D vector of theory predictions for all data points. Note that the prediction function is already

implemented; however, the user is allowed to override it in its own PDF application if the specific model needs extra features.

2.2 Likelihood function

Having defined a PDF model mapping $\mathbf{f} : \mathbb{R}^{N_\theta} \to \mathbb{R}^{N_{\text{fl}} \times N_x}$, `Colibri` requires a likelihood function $\mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta})$ that quantifies the agreement between theory predictions and experimental data. The likelihood is the fundamental ingredient underlying all inference methods implemented in our framework, namely Hessian, Monte Carlo, and Bayesian fits. Its specific form and implementation in `Colibri` are discussed in this section. Users may choose to employ the built-in implementation, with full control over its hyperparameters, or implement a custom version by overriding the `Colibri` likelihood function in their model whenever additional features are required.

Chi-Squared Likelihood
The basic form of the likelihood function is a chi-squared function:

$$\mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta}) = (\mathbf{D} - \text{FK}[\mathbf{f}(\boldsymbol{\theta})])^T \, C_{t_0}^{-1} \, (\mathbf{D} - \text{FK}[\mathbf{f}(\boldsymbol{\theta})]) \,, \quad (2.3)$$

where $\mathbf{D}$ is the data vector, $\text{FK}[\mathbf{f}(\boldsymbol{\theta})]$ is the forward map, and C_{t_0} is the t_0 covariance matrix customarily used in the NNPDF fits [46,47], with the t_0 prescription introduced to avoid the d'Agostini bias when data have multiplicative uncertainties [48]. During a fit, it is possible to impose positivity and integrability constraints on PDFs as well as on observables.

Positivity constraints
Probability distributions for physical observables must necessarily be non-negative quantities. PDFs beyond LO, however, are not probabilities, and thus they may be negative. Now, it was recently shown in Refs. [49,50] that, in the case of individual quark flavours and the gluon in the $\overline{\text{MS}}$ factorisation scheme, PDFs are indeed non-negative. We therefore allow users to impose this positivity condition along with the constraint of positivity of physical cross sections discussed below.

Positivity constraints on PDFs are implemented similarly to NNPDF4.0 [21], by adding extra Lagrange penalty terms to the likelihood function, namely

$$\mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta}) \;\longrightarrow\; \mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta}) \;+\; \Lambda_{\text{pos}} \sum_k \sum_i \text{Elu}_\alpha\big(-\tilde{f}_k(x_i, Q^2)\big) \,, \quad (2.4)$$

where by default $Q^2 = 5\,\text{GeV}^2$, Elu_α is the Exponential Linear Unit function with hyperparameter α, and the points x_i, following the NNPDF prescription, are ten values logarithmically spaced between 5×10^{-7} and 10^{-1}, plus ten points linearly spaced between 0.1 and 0.9. In addition, given that the positivity of $\overline{\text{MS}}$ PDFs is neither necessary nor sufficient in order to ensure cross section positivity, in order to exclude unphysical PDFs, we impose positivity of a number of cross sections, by means of Lagrange multipliers which penalise PDF configurations leading to negative physical observables. Specifically, positivity can be imposed on the F_2^u, F_2^d, F_2^s and F_L structure functions and of the flavour-diagonal Drell-Yan rapidity distributions $\sigma_{\text{DY}}^{u\bar{u}}$, $\sigma_{\text{DY}}^{d\bar{d}}$ and $\sigma_{\text{DY}}^{s\bar{s}}$. An important point to note is that the penalty parameter Λ_{pos} should not be regarded as a hyperparameter of the system. The reason for this is that the physically meaningful value of this hyperparameter is, in principle, known and should be infinitely large. In practice, however, setting Λ_{pos} too large introduces computational difficulties that slow down the convergence of the optimisation. For this reason, we typically choose the value of this penalty empirically, ensuring both convergence and the positivity of the resulting solution.

Integrability constraints
Integrability constraints are enforced similarly, by adding a term

$$\mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta}) \;\longrightarrow\; \mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta}) \;+\; \Lambda_{\text{int}} \sum_k \sum_i \big[x_i \, f_k(x_i, Q_0^2)\big]^2 \,, \quad (2.5)$$

where Q_0 is the parametrisation scale and the x_i run over the small-x region of the FK-table grid (in practice, often only the smallest x value is used to enforce this condition). The penalty term Λ_{int} is selected following the same empirical procedure as for the positivity penalty.

2.3 Data and theory predictions

`Colibri` provides a flexible platform that allows fitting PDF models to data that includes at least one incoming proton. The data is modelled in the framework of collinear QCD factorisation, where the scattering process is written as a convolution of the PDFs with perturbatively-computed, hard-scattering cross sections. In this context, inferring the PDFs from experimental measurements is an inverse problem; the unknowns are the PDFs, and the forward model consists of the hard-scattering cross section combined with PDF evolution kernels, commonly stored as FK-tables [51] (fast-kernel-tables). The data and FK-tables used in `Colibri` are inherited from the NNPDF framework [42].

We distinguish two classes of forward maps based on whether the initial state involves one proton (Deep Inelastic Scattering, DIS) or two protons (hadron–hadron collisions).

DIS data
DIS data is the most abundant data type in global PDF fits and is the most straightforward to model. For example, a

measurement of the F_2 structure function, consisting of $N_{\rm dat}$ points, can be written as the contraction of two operators:

$$F_{2,i} = \sum_{j=1}^{N_{\rm fl}} \sum_{k=1}^{N_x} {\rm FK}_{i,j,k}\ f_{j,k}\,, \tag{2.6}$$

where the operator ${\rm FK}_{i,j,k}$ has shape $(N_{\rm dat},\ N_{\rm fl},\ N_x)$, and $f_{j,k}$ is the $(N_{\rm fl} \times N_x)$-dimensional grid representing the PDF values at the input scale Q_0.

Hadron–hadron predictions
Hadron–hadron collisions are more complicated to model than DIS data, as they involve the convolution of two incoming partons, each with their own PDF. An $N_{\rm dat}$-point measurement σ of a hadron–hadron cross section can be written as

$$\sigma_i = \sum_{j=1}^{N_{\rm fl}} \sum_{k=1}^{N_{\rm fl}} \sum_{l=1}^{N_x} \sum_{m=1}^{N_x} {\rm FK}_{i,j,k,l,m}\ f_{j,l}\ f_{k,m}\,, \tag{2.7}$$

where the operator ${\rm FK}_{i,j,k,l,m}$ has shape $(N_{\rm dat},\ N_{\rm fl},\ N_{\rm fl},\ N_x,\ N_x)$.

2.4 Inference methods

In its release version, `Colibri` provides four inference strategies:

– **Analytic fit**: computes the posterior mean and covariance of the parameters from the closed-form solution of a linear regression problem, yielding a Gaussian posterior.
– **Hessian method**: estimates parameter uncertainties from the curvature of the likelihood around the best-fit point, effectively linearising the problem in the vicinity of the minimum.
– **Monte Carlo replica method** [20]: constructs an ensemble of pseudodata replicas that reproduce experimental uncertainties, fits each independently, and uses the statistical spread to approximate the posterior.
– **Bayesian inference**: explores the full posterior distribution of the parameters using modern nested-sampling algorithms, providing a principled probabilistic treatment of uncertainties.

The following subsections examine each method in more detail, highlighting their assumptions, strengths, and limitations.

2.4.1 Analytic fits

The analytic fit is the closed-form solution of a linear regression problem with Gaussian errors. It applies only when both the PDF model and the forward map are linear in the parameters, so that the likelihood is strictly quadratic. With a uniform prior on the parameters (and hence on the PDF values), the posterior is Gaussian, and its mean and covariance can be obtained directly from closed-form expressions, without any iterative optimisation or sampling.

This method cannot accommodate non-linear constraints such as positivity or integrability, and is therefore not suitable for realistic global PDF determinations. Nevertheless, it remains useful in several contexts:

– **Fast benchmarks:** provides a lightweight means to validate new PDF parametrisations or data subsets before running a full fit.
– **Bayesian updating:** if a subset of data satisfies the linearity conditions, the resulting Gaussian posterior from that fit can serve as a prior in a subsequent fit to an uncorrelated dataset, thereby reducing the dimensionality of the sampling problem (see Appendix A for more details).
– **Cross-checks:** analytic fits allow one to test the consistency of linear approximations against more general inference strategies, highlighting the impact of non-linearities.

In practice, if the PDF model is linear in the parameters, the analytic fit is particularly useful for fitting linear DIS observables without constraints, where it yields closed-form posterior distributions at negligible computational cost. A detailed mathematical illustration of the method is provided in Appendix B.

2.4.2 Hessian method

The Hessian method is a widely used approach for parameter inference in PDF fits. It is based on a quadratic approximation of the likelihood around its minimum, so that uncertainties can be propagated through the covariance matrix of the fit parameters.

In `Colibri`, the best-fit point is obtained by minimising the likelihood function using gradient-based optimisation algorithms provided by the `Optax` library [52]. Once the minimum is found, the Hessian matrix H of second derivatives with respect to the parameters is computed, and its inverse provides the covariance matrix C. This covariance encodes the parameter uncertainties under the assumption of a locally Gaussian likelihood.

As in traditional PDF analyses, it is possible to introduce a *tolerance* factor $T > 1$ to account for tensions or inconsistencies among datasets. In this case, the covariance matrix is rescaled as

$$C \longrightarrow T^2 C\,, \tag{2.8}$$

thus affecting the quoted parameter uncertainties [53]. There are more modern ways to implement a dynamic tolerance, rather than a global one, that could be implemented in `Colibri`, see [9,12,54] for detailed discussions.

For practical applications, Hessian PDF sets are distributed in terms of eigenvector directions of the covariance matrix. Diagonalising C yields a set of eigenvalues and eigenvectors,

$$C = V \Lambda V^T, \tag{2.9}$$

which define orthogonal directions in parameter space. The corresponding error sets are obtained by shifting the best-fit parameters along each eigenvector direction by $\pm\sqrt{\lambda_i}$, where λ_i is the i-th eigenvalue. This representation is the standard format used in global PDF analyses, and is directly supported by `Colibri`.

The Hessian method is computationally efficient and provides a compact representation of PDF uncertainties. However, it relies on the quadratic approximation of the likelihood and may result in unfaithful uncertainties in the presence of non-linearities or parameter degeneracies. In practice, the current implementation breaks down in the presence of degeneracies, since the Hessian matrix is not of full rank and the corresponding covariance matrix cannot be constructed. This problem commonly occurs for highly parameterised or redundant models. In addition, the numerical stability of the Hessian diagonalisation can become a limiting factor as the dimensionality of the parametrisation increases. In practice, ill-conditioned Hessian matrices may lead to unstable eigenvalue spectra and unreliable error propagation, requiring regularisation or dimensionality reduction techniques to ensure a stable inversion. In `Colibri`, it serves both as a benchmark against which more general inference strategies can be compared, and as a practical tool for generating traditional Hessian PDF sets.

2.4.3 Monte Carlo replica method

The Monte Carlo (MC) replica method is a widely used approach to estimate PDF uncertainties through repeated fits to pseudodata samples. In this approach, one generates N_{rep} replicas of the experimental dataset by sampling from a multivariate normal distribution with mean given by the central data values and covariance equal to the experimental covariance matrix. Each replica is then fitted independently, typically by minimising the chi-squared function using gradient-based optimisation algorithms, which in `Colibri` are provided by JAX [43] and Optax [52]. The resulting ensemble of best-fit parameter sets approximately draws from the posterior distribution of the model parameters, and does so exactly in a linear regime. PDF uncertainties are then obtained from the statistical spread of this ensemble.

The MC replica method has several practical advantages:

- **Robustness:** it makes minimal assumptions about the form of the likelihood surface, and does not require explicit computation of the Hessian matrix.
- **Flexibility:** it naturally incorporates correlations among data points and can accommodate non-linear models at the fitting stage.
- **Interpretability:** the replica ensemble provides a transparent representation of uncertainties that can be propagated to observables without additional approximations.

However, the method also has important limitations. As shown in Ref. [20], the MC replica ensemble is formally equivalent to Bayesian posterior samples only in the case of linear models with Gaussian likelihoods. For non-linear parametrisations or forward maps, it can introduce biases and produce unreliable uncertainty estimates. Moreover, the need to perform N_{rep} full fits makes the approach computationally more expensive than Hessian methods, especially for complex models.

For these reasons, `Colibri` implements the replica method primarily for benchmarking and for compatibility with existing PDF fitting practices, while Bayesian nested sampling (Sect. 2.4.4) is recommended as the principled approach for fully non-linear problems.

2.4.4 Bayesian inference

Bayesian inference is the recommended strategy in `Colibri` for realistic PDF determinations. It provides a principled statistical foundation for uncertainty quantification, naturally incorporates prior knowledge, and enables robust model comparison. In this framework, the goal of a PDF fit is to characterise the full posterior distribution of the model parameters $\boldsymbol{\theta}$,

$$p(\boldsymbol{\theta} \mid \mathbf{D}) \;\propto\; \mathcal{L}(\mathbf{D} \mid \boldsymbol{\theta})\, \pi(\boldsymbol{\theta})\,, \tag{2.10}$$

where $\mathcal{L}$ is the likelihood function and π is the prior. The posterior encodes all information about the parameters given the data, allowing for a fully probabilistic treatment of uncertainties and correlations. Unlike the Hessian or MC replica methods, it does not rely on linear approximations or pseudodata ensembles, and is therefore valid for arbitrary parametrisations and forward models.

A core strength of `Colibri` is the combination of user-defined PDF parametrisations with Bayesian inference. Through nested-sampling algorithms, this not only yields robust uncertainty quantification, but also provides the Bayesian evidence required for principled model selection and systematic comparisons of alternative parametrisations.

Implementation in `Colibri`

Sampling the posterior of high-dimensional PDF models is a challenging computational problem. `Colibri` addresses this using modern nested-sampling algorithms, as implemented in the `UltraNest` package [55]. Nested sampling is well suited to PDF fits because it efficiently explores parameter spaces that may be multi-modal or strongly correlated, while also providing an estimate of the Bayesian evidence for principled model comparison. This makes Bayesian inference not only the most robust option for uncertainty quantification, but also a powerful tool for testing alternative PDF parametrisations within the same framework. The potential of this approach has already been demonstrated in our recent work [33], where dimensionality reduction was applied to the NNPDF neural network parametrisation. That study delivered the first realistic Bayesian PDF fit using `Colibri` and showed that model selection can be performed with minimal parametrisation complexity while maintaining excellent agreement with data.

Prior distributions

Bayesian inference requires the specification of a prior $\pi(\boldsymbol{\theta})$, which encodes information about the parameters before the data are taken into account. `Colibri`, at the time of the release, supports two built-in options:

- **Uniform priors**, with configurable bounds for each parameter.
- **Gaussian priors**, defined by a mean vector and covariance matrix taken from the posterior of a previous fit.

The Gaussian option enables a Bayesian update (posterior–factorisation): when an earlier fit yields an approximately Gaussian posterior and the datasets are uncorrelated, that posterior can be re-used as the prior for a subsequent fit. Appendix A discusses the theoretical basis and domain of validity of this approach.

In addition, users may implement fully customised priors by overriding the prior function in their model definition. This flexibility is essential for incorporating external information or theoretical constraints, and exemplifies the modularity of the `Colibri` framework. See Code Listing 2 for an example of how to override the `Colibri` built-in prior to specify a unit gaussian prior for the parameters.

```python
import jax
from colibri import bayes_prior
from colibri.utils import cast_to_numpy

def bayesian_prior(prior_settings, pdf_model):
    """
    Override of bayesian_prior:
    - if prior_distribution == "unit_gaussian", returns a unit Gaussian prior
    - otherwise falls back to colibri.bayes_prior
    """

    if prior_settings.prior_distribution == "unit_gaussian":

        @cast_to_numpy
        @jax.jit
        def prior_transform(cube):
            # transform [0,1] -> N(0,1)
            return jax.scipy.stats.norm.ppf(cube)

        return prior_transform

    # fallback to the normal bayesian_prior
    return bayes_prior.bayesian_prior(prior_settings, pdf_model)
```

Code Listing 2 Example of how to override the prior built-in `Colibri`, specifying a unit gaussian prior for the parameters instead.

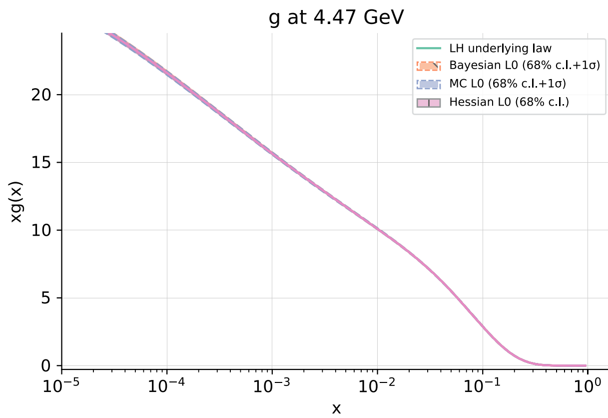
3 Case study: a simple `Colibri` fit with Les Houches PDFs

To demonstrate the capabilities of `Colibri`, we present a set of benchmark fits performed with a simple model that was put forward in one of the Les Houches benchmarks to compare various PDF fitting methodologies [56]. The Les Houches model provides a simple polynomial parametrisation of PDFs under some assumptions that are explicitly spelled out in Appendix D. In the evolution basis, the four independent PDFs are parametrised as:

$$
\begin{aligned}
xf_g(x,Q_0) &= A_g\, x^{\alpha_g}\,(1-x)^{\beta_g}\\
xf_\Sigma(x,Q_0) &= A_\Sigma\, x^{\alpha_\Sigma}\,(1-x)^{\beta_\Sigma}\\
xf_V &= xf_{u_v}+xf_{d_v}\\
&= A_{u_v}\, x^{\alpha_{u_v}}\,(1-x)^{\beta_{u_v}}\,(1+\epsilon_{u_v}\sqrt{x}+\gamma_{u_v}x)\\
&\quad + A_{d_v}\, x^{\alpha_{d_v}}\,(1-x)^{\beta_{d_v}}(1+\epsilon_{d_v}\sqrt{x}+\gamma_{d_v}x)\\
xf_{V_3} &= xf_{u_v}-xf_{d_v}\\
&= A_{u_v}\, x^{\alpha_{u_v}}\,(1-x)^{\beta_{u_v}}\,(1+\epsilon_{u_v}\sqrt{x}+\gamma_{u_v}x)\\
&\quad - A_{d_v}\, x^{\alpha_{d_v}}\,(1-x)^{\beta_{d_v}}(1+\epsilon_{d_v}\sqrt{x}+\gamma_{d_v}x).
\end{aligned}
\tag{3.1}
$$

After applying the Les Houches parametrisation assumptions and sum rules (spelled out explicitly in Appendix D, along with a detailed discussion of the rotation to the evolution basis and the expressions for the normalisation factor) we are left with 13 free parameters, namely $\alpha_g,\ \beta_g,\ \alpha_{u_v},\ \beta_{u_v},\ \epsilon_{u_v},\ \gamma_{u_v},\ \alpha_{d_v},\ \beta_{d_v},\ \epsilon_{d_v},\ \gamma_{d_v},\ \alpha_\Sigma,\ \beta_\Sigma$ and $A_g(A_\Sigma)$.

To showcase the performance of the new tool, we consider a fit to synthetic data, as performed in Refs. [1,57]. There are



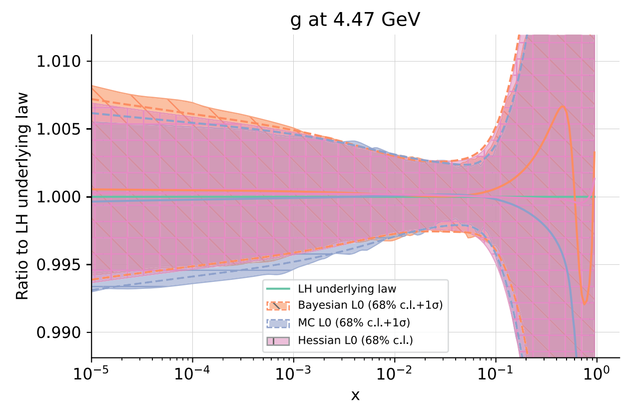
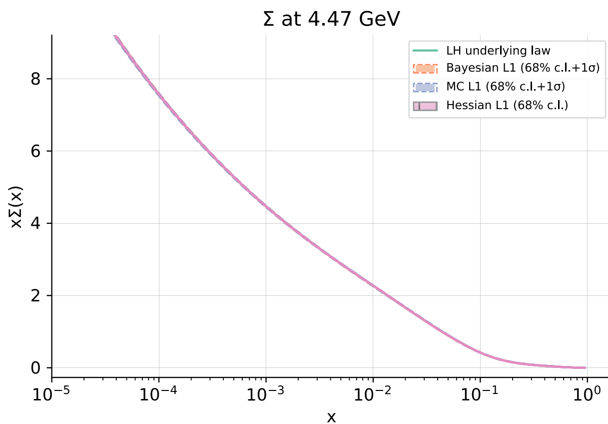


Fig. 2 A gluon PDF fit resulting from a level 0 closure test computed with the Monte Carlo replica method (blue), Bayesian inference (orange), and the Hessian method (red). The green line shows the underlying law to be recovered, which in this case is the Les Houches parametrisation with best-fit parameter values from Ref. [56]. The right-hand panel shows the ratio to the underlying law



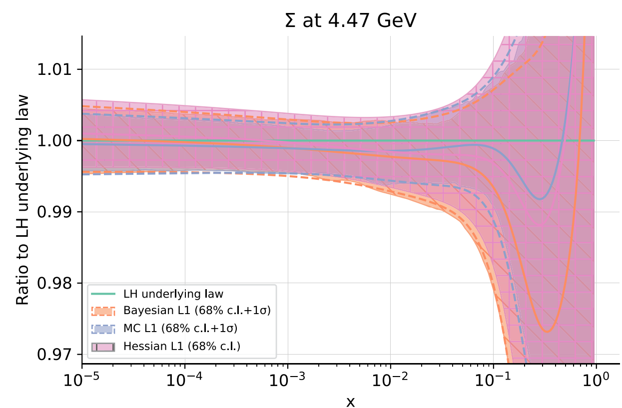


Fig. 3 A Σ PDF fit resulting from a level 1 closure test computed with the Monte Carlo replica method (blue), Bayesian inference (orange), and the Hessian method (pink). The green line shows the underlying law to be recovered, which in this case is the Les Houches parametrisation with best-fit parameter values from Ref. [56]. The right-hand panel shows the ratio to the underlying law

two such levels of data, namely

$$\mathbf{D}^{L_0} = \mathbf{T}[\mathbf{f}(\boldsymbol{\theta}^*)], \tag{3.2}$$

where $\boldsymbol{\theta}^*$ are the "true" PDF parameters, taken from some underlying law (in this case the best fit parameters determined in [56]) that are used to generate the theory predictions for observables. This is called Level-0 data, and is nothing but the underlying law itself, built by convolving the "true" PDFs with partonic cross sections computed at a given perturbative order. Adding Gaussian noise generated from the covariance matrix of the input data, we obtain Level-1 data, namely;

$$\mathbf{D}^{L_1} = \mathbf{T}[\mathbf{f}(\boldsymbol{\theta}^*)] + \eta, \tag{3.3}$$

where $\eta \sim \mathcal{N}(0, C)$, and C is the covariance matrix used in the fit. We fit it only to the set of DIS data included in the NNPDF4.0 analysis [21]. In a Level-0 test, synthetic data are generated directly from the underlying parametrisation without statistical fluctuations, so the goal of the fit is to recover the exact law. In a Level-1 test, synthetic data include statistical noise consistent with the experimental covariance, making the exercise closer to a realistic fit.

Figures 2 and 3 show representative results for the gluon and Σ PDFs. In all cases, the underlying law (green) is well reproduced by Bayesian (orange), Monte Carlo replica (blue), and Hessian (pink) fits. As expected, the Level-0 closure test demonstrates near-perfect agreement with the generating function, while in the Level-1 test the fitted distributions track the law within the quoted uncertainties. The right-hand

Table 1 Comparison of Bayesian, Monte Carlo and Hessian χ^2 values for Level 0 and Level 1 closure tests, where the underlying law was the Les Houches parametrisation model

	Bayesian	MC	Hessian
Level 0	5.21×10^{-4}	1.76×10^{-5}	2.39×10^{-5}
Level 1	1.00	1.01	1.01

panels display the ratio to the "truth", highlighting the consistency of the three approaches across the full x range. In this simplified scenario the three methodologies yield comparable results, likely because the Gaussian approximation holds well. In more general settings, however, this need not be the case, and deviations between inference methods may occur.

Table 1 compares the χ^2 values obtained with the Bayesian, Monte Carlo, and Hessian methods. All three strategies give compatible results in both closure tests, confirming the internal consistency of the framework. Note that in a Level 0 closure test, the underlying law is generated from the same model used for the fit, leading to an expected χ^2 of 0. In contrast, in a Level 1 test, statistical fluctuations are introduced, and the expected χ^2 rises to 1.

An additional advantage of the Bayesian approach, i.e., explicitly sampling the posterior distribution of the PDF parameters for which a prior has been defined, is that it yields direct samples from the posterior distribution of the PDF parameters. Beyond the reduced set of replicas exported in LHAPDF format, `Colibri` retains the full collection of posterior samples generated during the nested-sampling run. These samples can be analysed further to extract detailed information about the parameter space, for instance by producing corner plots that expose correlations and degeneracies among parameters. Both the Hessian and Monte Carlo approaches allow one to estimate correlations between parameters. However, their accuracy is optimal when the posterior distribution is approximately Gaussian, that is within the linear regime around the best-fit point. Bayesian sampling methods, on the other hand, can robustly capture non-linear correlations and multi-modal structures, providing a more faithful representation of the full parameter landscape. Figure 4 shows an example corner plot from the Bayesian Level-1 closure test, illustrating how `Colibri` facilitates a transparent exploration of the multidimensional posterior beyond the one-dimensional PDF projections.

These benchmarks illustrate how `Colibri` makes it straightforward to perform and compare fits with different inference strategies within a single infrastructure. By applying them to the same dataset and parametrisation, one can systematically study the assumptions and limitations of each method and validate the robustness of PDF determinations.

4 Conclusions

In this work we have presented `Colibri`, a new open-source platform for parton distribution function determinations. The framework is designed around two central principles: user-defined PDF parametrisations and a unified treatment of inference. This modularity allows for direct comparisons between inference strategies, Hessian, Monte Carlo replicas, and Bayesian nested sampling, under identical conditions, thereby exposing their respective strengths and limitations. As illustrated in Appendix C, we have shown how a PDF model can be implemented in `Colibri` by using the Les Houches parametrisation as a worked example. With this model, we performed closure tests that showcase the results of the three inference strategies available in the framework. These tests confirm that `Colibri` reproduces the expected behaviour of all methods, while highlighting the advantages of Bayesian inference as a principled and fully probabilistic characterisation of uncertainties. In addition, the ability to exploit posterior samples directly for correlation studies further demonstrates the flexibility and power of the Bayesian approach within the `Colibri` framework.

`Colibri` is intended as a living project. The code is under active development, with new features and inference strategies continuously being added. In addition to the core infrastructure, we also plan to provide ready-to-use PDF models as part of the distribution. One such example is already available from our recent study [33], which provides an implementation of a model that performs dimensional reduction of a neural network into its dominant modes, available at https://github.com/HEP-PBSP/wmin-model. Other models that are being developed are models based on Gaussian Processes [34], models based on more realistic polynomial parametrisations and models based on neural networks. Users are encouraged to implement more models, e. g. those based on flexible polynomial basis [58,59] or advanced ML tools such as Bayesian neural networks. The online documentation is continuously updated and serves as the primary reference for recent progress, available functionalities, and tutorials.

In the near future, `Colibri` will be extended to fit PDFs simultaneously with SM precision parameters as well as with the Wilson Coefficients that parametrise New Physics degrees of freedom in some Effective Field Theories. Such a step is crucial, as the correlation between PDFs and SM precision parameters can no longer be ignored [60–65], and the availability of open-source tools for simultaneous fits of PDFs and SM parameters would be a key advancement in keeping correlations into account. In particular, while precise LHC data significantly enhance PDF precision, it has been shown in several recent publications [66–73] that they can also be sensitive to BSM dynamics. If BSM signals distort an experimental distribution included in a PDF fit, which is typically assumed to follow the SM, this can lead to incon-

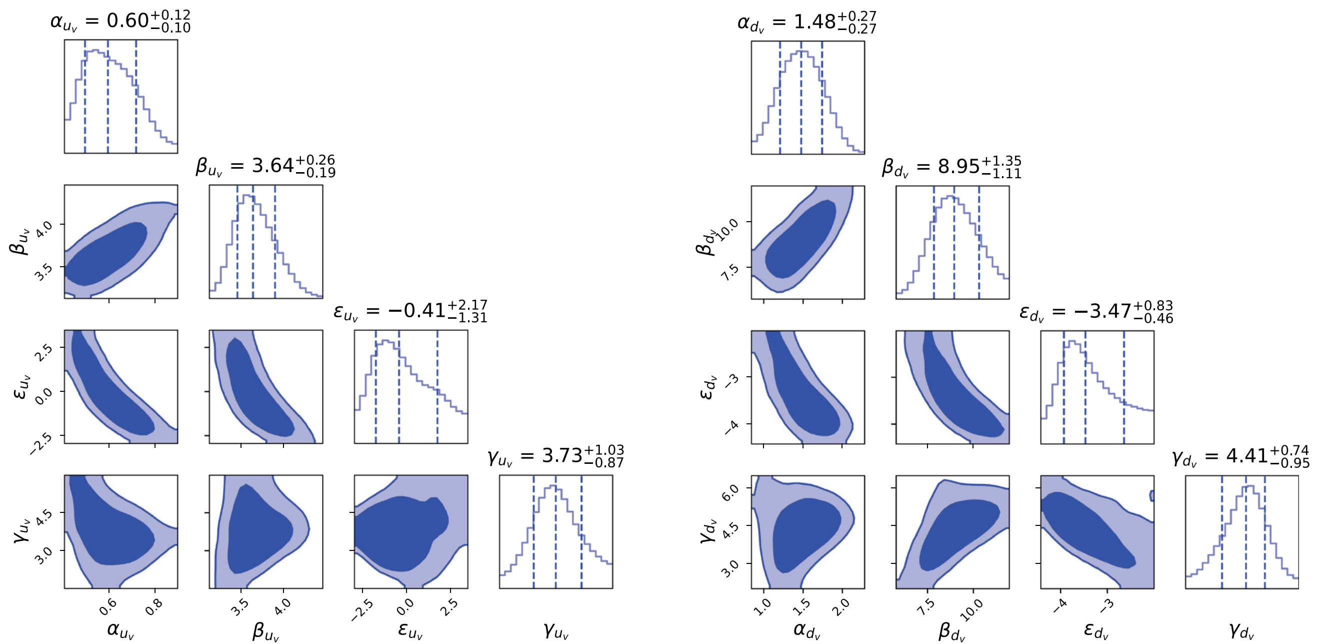


Fig. 4 Example corner plots of posterior samples. The left panel shows the parameters associated with the u valence quark, and the right panel those of the d valence quark. In principle, analogous plots could be produced for all 13 parameters, but here we restrict to subsets for clarity and illustration

sistencies, which in turn can have PDFs adopting and incorporating the BSM effects, resulting in BSM-biased PDFs. A flexible tool for a simultaneous fit of PDFs and Wilson coefficients, in the same spirit but more general as compared to those provided by `SimuNET` [72] or `xFitter` [74], would avoid such bias and ensure that BSM effects and PDF effects can be disentangled. The extension of `Colibri` in both directions would provide the flexible tool that is needed to explore this completely new territory with a Bayesian inference method that allows full control over the role of prior assumptions.

The `Colibri` code is publicly available from its GitHub repository:

https://github.com/HEP-PBSP/colibri,

and is accompanied by documentation and tutorials provided at:

https://hep-pbsp.github.io/colibri/.

Acknowledgements We thank Gaia Fontana (`@qftoons`) for designing the `Colibri` logo. We thank Ella Cole, Francesco Merlotti, Elie Hammou, Manuel Moreales Alvarado, David Yallup and the members of the NNPDF collaboration for insightful discussions. We are indebted to Juan Cruz Martinez for his help with the NNPDF code, and Zahari Kassabov for his contributions during the earliest stages of the project. Mark N. Costantini and Maria Ubiali are supported by the European Research Council under the European Union's Horizon 2020 research and innovation Programme (Grant agreement no. 950246), and partially by the STFC consolidated Grant ST/X000664/1. LM acknowledges support from the European Union under the MSCA fellowship (Grant agreement no. 101149078) *Advancing global SMEFT fits in the LHC precision era (EFT4ward)*. J. M. M. is supported by the donation of Christina and Peter Dawson to Lucy Cavendish College. Valentina Schütze Sánchez is supported by the Newnham Scholarship for Women in Theoretical Physics.

Data Availability Statement This manuscript has no associated data. [Author's' comment: Data sharing not applicable to this article as no datasets were generated or analysed during the current study.]

Code Availability Statement This manuscript has associated code/-software in a data repository. [Author's comment: Available in Colibri's GitHub repository: https://github.com/HEP-PBSP/colibri.]

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.
Funded by SCOAP[3].

Appendix A: Bayesian update

Suppose experimental data comprising N_{data} datapoints is distributed according to a multivariate normal:

$$\mathbf{D} \sim \mathcal{N}\big(\text{FK}(\boldsymbol{\theta}),\ \Sigma\big),$$

where Σ is the covariance matrix of dimension $N_{\text{data}} \times N_{\text{data}}$.

In Bayesian statistics, $\boldsymbol{\theta}$ itself is treated as a random variable with prior density $\pi(\boldsymbol{\theta})$, here taken to be a sufficiently wide uniform distribution. After observing $\mathbf{D}_0$, Bayes' theorem yields the posterior

$$p(\boldsymbol{\theta} \mid \mathbf{D}_0) = \frac{\pi(\boldsymbol{\theta})\, L(\mathbf{D}_0 \mid \boldsymbol{\theta})}{Z} = \frac{\pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2} \|\mathbf{D}_0 - \text{FK}(\boldsymbol{\theta})\|^2_{\Sigma}\right)}{\int d\boldsymbol{\theta}\, \pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2} \|\mathbf{D}_0 - \text{FK}(\boldsymbol{\theta})\|^2_{\Sigma}\right)}. \quad \text{(A.1)}$$

where we define the generalised L_2 norm

$$\|\mathbf{x}\|^2_{\Sigma} = \mathbf{x}^T\, \Sigma^{-1}\, \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^{N_{\text{data}}}.$$

Now assume $\mathbf{D}_0 = (\mathbf{D}_1, \mathbf{D}_2)^T$ with $\mathbf{D}_1 \in \mathbb{R}^{n_1}, \mathbf{D}_2 \in \mathbb{R}^{n_2}$, $n_1 + n_2 = N_{\text{data}}$, and that the two subsets are uncorrelated so that

$$\Sigma = \Sigma_1 \oplus \Sigma_2, \quad \Sigma_1 \in \mathbb{R}^{n_1 \times n_1}, \; \Sigma_2 \in \mathbb{R}^{n_2 \times n_2}.$$

The likelihood then factorises, and from (A.1) we obtain

$$p(\boldsymbol{\theta} \mid \mathbf{D}_0) = \frac{\pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2}\|\mathbf{D}_1 - \text{FK}_1(\boldsymbol{\theta})\|^2_{\Sigma_1}\right) \exp\left(-\frac{1}{2}\|\mathbf{D}_2 - \text{FK}_2(\boldsymbol{\theta})\|^2_{\Sigma_2}\right)}{\displaystyle\int d\boldsymbol{\theta}\, \pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2}\|\mathbf{D}_1 - \text{FK}_1(\boldsymbol{\theta})\|^2_{\Sigma_1}\right) \exp\left(-\frac{1}{2}\|\mathbf{D}_2 - \text{FK}_2(\boldsymbol{\theta})\|^2_{\Sigma_2}\right)}, \quad \text{(A.2)}$$

where $\text{FK}(\boldsymbol{\theta}) = (\text{FK}_1(\boldsymbol{\theta}), \text{FK}_2(\boldsymbol{\theta}))^T$.

Next, note that the posterior given only $\mathbf{D}_1$ is

$$p_{\mathbf{D}_1}(\boldsymbol{\theta} \mid \mathbf{D}_1) = \frac{\pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2}\|\mathbf{D}_1 - \text{FK}_1(\boldsymbol{\theta})\|^2_{\Sigma_1}\right)}{\displaystyle\int d\boldsymbol{\theta}\, \pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2}\|\mathbf{D}_1 - \text{FK}_1(\boldsymbol{\theta})\|^2_{\Sigma_1}\right)} = \frac{\pi(\boldsymbol{\theta}) \exp\left(-\frac{1}{2}\|\mathbf{D}_1 - \text{FK}_1(\boldsymbol{\theta})\|^2_{\Sigma_1}\right)}{Z_1}. \quad \text{(A.3)}$$

Substituting into (A.2) gives

$$p(\boldsymbol{\theta} \mid \mathbf{D}_0) = \frac{p_{\mathbf{D}_1}(\boldsymbol{\theta} \mid \mathbf{D}_1) \exp\left(-\frac{1}{2}\|\mathbf{D}_2 - \text{FK}_2(\boldsymbol{\theta})\|^2_{\Sigma_2}\right)}{\displaystyle\int d\boldsymbol{\theta}\; p_{\mathbf{D}_1}(\boldsymbol{\theta} \mid \mathbf{D}_1) \exp\left(-\frac{1}{2}\|\mathbf{D}_2 - \text{FK}_2(\boldsymbol{\theta})\|^2_{\Sigma_2}\right)}.$$

If $\mathbf{D} \sim \mathcal{N}(\text{FK}(\boldsymbol{\theta}), \Sigma)$ with $\Sigma = \bigoplus_{i=1}^n \Sigma_i$, this argument applies recursively, yielding

$$p(\boldsymbol{\theta} \mid \mathbf{D}_0) = \frac{\prod_{i=1}^{n-1} p_{\mathbf{D}_i}(\boldsymbol{\theta} \mid \mathbf{D}_i) \exp\left(-\frac{1}{2}\|\mathbf{D}_n - \text{FK}_n(\boldsymbol{\theta})\|^2_{\Sigma_n}\right)}{\int d\boldsymbol{\theta}\; \prod_{i=1}^{n-1} p_{\mathbf{D}_i}(\boldsymbol{\theta} \mid \mathbf{D}_i) \exp\left(-\frac{1}{2}\|\mathbf{D}_n - \text{FK}_n(\boldsymbol{\theta})\|^2_{\Sigma_n}\right)},$$

with

$$p_{\mathbf{D}_k}(\boldsymbol{\theta} \mid \mathbf{D}_k) = \frac{\prod_{i=1}^{k-1} p_{\mathbf{D}_i}(\boldsymbol{\theta} \mid \mathbf{D}_i) \exp\left(-\frac{1}{2}\|\mathbf{D}_k - \text{FK}_k(\boldsymbol{\theta})\|^2_{\Sigma_k}\right)}{\int d\boldsymbol{\theta}\; \prod_{i=1}^{k-1} p_{\mathbf{D}_i}(\boldsymbol{\theta} \mid \mathbf{D}_i) \exp\left(-\frac{1}{2}\|\mathbf{D}_k - \text{FK}_k(\boldsymbol{\theta})\|^2_{\Sigma_k}\right)}.$$

Appendix B: Bayesian linear regression

To illustrate the analytic method, let us assume a likelihood of the kind

$$p(D \mid \theta) = \frac{1}{(2\pi)^{N_{\text{dat}}/2}\, |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\,(D - f(\theta))^T\, \Sigma^{-1}\, (D - f(\theta))\right), \quad \text{(B.1)}$$

with θ being the parameters of the model, D the data vector and Σ its covariance. In general, a linear model can be described by the following equation:

$$f(\theta) = W\,\theta, \quad \text{(B.2)}$$

where W is a matrix that maps θ to the theory prediction vector, $f(\theta)$. For such a model, the likelihood factorises as:

$$p(D \mid \theta) = \frac{(2\pi)^{N_\theta/2}\, \left|(W^T\, \Sigma^{-1} W)^{-1}\right|^{1/2}}{(2\pi)^{N_{\text{dat}}/2}\, |\Sigma|^{1/2}} \times \exp\left(-\frac{1}{2}\,(D - \hat{D})^T\, \Sigma^{-1}\, (D - \hat{D})\right) \times \frac{\exp\left(-\frac{1}{2}\,(\theta - \hat{\theta})^T\, W^T\, \Sigma^{-1} W\, (\theta - \hat{\theta})\right)}{(2\pi)^{N_\theta/2}\, \left|(W^T\, \Sigma^{-1} W)^{-1}\right|^{1/2}}$$
$$= (2\pi)^{N_\theta/2}\, \left|(W^T\, \Sigma^{-1} W)^{-1}\right|^{1/2}\, p(D \mid \hat{\theta})\, p(\hat{\theta} \mid \theta), \quad \text{(B.3)}$$

where N_θ is the number of parameters in the model, and we have defined the following relations;

$$\hat{\theta} = (W^T\, \Sigma^{-1}\, W)^{-1}\, W^T\, \Sigma^{-1}\, D, \quad \hat{D} = W\,\hat{\theta}, \quad \text{(B.4)}$$

and

$$p(\hat{\theta} \mid \theta) = \frac{\exp\left(-\frac{1}{2}\,(\theta - \hat{\theta})^T\, W^T\, \Sigma^{-1} W\, (\theta - \hat{\theta})\right)}{(2\pi)^{N_\theta/2}\, \left|(W^T\, \Sigma^{-1} W)^{-1}\right|^{1/2}}. \quad \text{(B.5)}$$

With a uniform prior,

$$p(\theta_i) = \begin{cases} \dfrac{1}{b_i - a_i}, & \theta_i \in [a_i, b_i], \\ 0, & \text{otherwise}, \end{cases} \quad \text{(B.6)}$$

the posterior becomes

$$p(\theta \mid D) \propto p(D \mid \theta)\, p(\theta)$$

$$\propto p(D \mid \theta) \prod_{i=1}^{N_\theta} \frac{\Theta(\theta_i - a_i)\,\Theta(b_i - \theta_i)}{b_i - a_i}. \quad \text{(B.7)}$$

Appendix C: How to implement a PDF model in `Colibri`

This appendix presents an example of how to implement a PDF model in `Colibri`. We will first show how to download the code, and then present an example of a model implementation, discussing the building blocks of a `Colibri` model. We will then show how to implement the Les Houches parametrisation as an example.

The content presented in this appendix is discussed in further detail in the `Colibri` documentation, https://hep-pbsp.github.io/colibri/.

Appendix C.1: Installing `Colibri` on Linux or macOS

This section covers installing `Colibri` in various ways.

Appendix C.1.1: Development installation via Conda

You can install colibri easily by first cloning the repository and then using the provided `environment.yml` file:

```
git clone https://github.com/HEP-PBSP/colibri
cd colibri
```

From your conda base environment run:

```
conda env create -f environment.yml
```

This will create a `colibri-dev` environment installed in development mode. If you want to use a different environment name you can run:

```
conda env create -n myenv -f environment.yml
```

Appendix C.1.2: Installing with pip

If you don't want to clone the repository and don't need to work in development mode you can follow the installation instructions below.

Note that most of the `Colibri` dependencies are available in the PyPi repository. However non-python codes such as LHAPDF and pandoc won't be installed automatically and need to be manually installed in the environment. Because of this, we recommend the use of a conda environment. So the first step would be to create one from your base environment. For instance;

```
conda create -n colibri-dev python>=3.11
```

In this new environment, install the following conda packages:

```
conda install mpich lhapdf pandoc mpi4py ultranest pip
```

After having completed this, you can simply install the rest of the dependencies with `pip`:

```
python -m pip install git+https://github.com/HEP-PBSP/colibri.git
```

Note that this will install the latest development version. If you want to install a specific release, you can specify the version. For instance, for v1.0.0, you can use the following command:

```
python -m pip install git+https://github.com/HEP-PBSP/colibri.git@v1.0.0
```

To verify that the installation went through:

```
python -c "import colibri; print(colibri.__version__)"
colibri --help
```

Appendix C.1.3: GPU (CUDA) JAX support

The installation instructions shown above will install JAX in cpu mode. It is however possible to run `Colibri` fits using GPU cuda support too. To do so, after installing the package following one of the methods shown above, if you are on a Linux machine you can install JAX in CUDA mode by running:

```
pip install -U "jax[cuda12]" -f https://storage.googleapis.com/jax-releases/jax_releases.html
```

(Note that this is a single command line).

It is possible to run fits using float32 precision. The only way of doing so currently is to apply a patch to UltraNest so that the `json.dump` is compatible. To do that, run the following commands;

```
git clone git@github.com:LucaMantani/UltraNest.git
cd UltraNest
git switch add-numpy-encoder
pip install .
```

Appendix C.2: Implementing a model in `Colibri`

In general, a `Colibri` model is contained in a directory with the following structure:

```
model_to_implement/
|-- pyproject.toml # Defines a python package
  for the project and sets up executable
|-- model_to_implement/
|   |-- app.py    # Enables the use of
  reportengine and validphys
|   |-- config.py# Defines the configur
    ation layer for the model
|   |-- model.py # Script where the
    model is defined
|-- runcards/    # Directory containing
   any runcards
```

The best way to understand how to implement a model is to go through an example, so let's have a look at how the Les Houches parametrisation is built.

Appendix C.3: Example: Les Houches parametrisation model

In this section, we discuss how to implement a model in `Colibri`, using the Les Houches parametrisation model as an example. This parametrisation is simple enough for us to exemplify the use of `Colibri`, while still being realistic enough that this tutorial can be used as a template for other, more complex parametrisations or models.

Following this parametrisation, our basis has four PDFs, which in the evolution basis, read as in Eq. (3.1). After applying the Les Houches parametrisation assumptions and sum rules spelled out in 1 we are left with 13 free parameters to fit, namely α_g, β_g, α_{u_v}, β_{u_v}, ϵ_{u_v}, γ_{u_v}, α_{d_v}, β_{d_v}, ϵ_{d_v}, γ_{d_v}, α_Σ, β_Σ and $A_g(A_\Sigma)$. We will now discuss how to implement this parametrisation in `Colibri`.

Appendix C.3.1: Implementing the Les Houches model in Colibri

In the `colibri/examples/` directory, you will find a directory called `les_houches_example`, which follows the structure defined above. We will have a look at them one by one.

`pyproject.toml`

The `pyproject.toml` file defines the Python package configuration for this model using Poetry[75] as the dependency management and packaging tool. The configuration file structure looks like this:

```
1
2  [build-system]
3  requires = ["poetry-core>=1.0.0", "
      poetry-dynamic-versioning>=1.1.0
      "]
4  build-backend = "
      poetry_dynamic_versioning.
      backend"
5
6  [tool.poetry]
7  name = "les_houches_example"
8  version = "1.0.0"
9  authors = ["PBSP collaboration"]
10 description = "Les Houches
      Parametrisation Example"
11
12 [tool.poetry.dependencies]
13
14
15 [tool.poetry.extras]
16 test = [
17     "pytest",
18     "hypothesis",
19 ]
20 doc = [
21     "sphinx",
22     "recommonmark",
23     "sphinx_rtd_theme"
24 ]
25
26 [tool.poetry.scripts]
27 les_houches_exe = "
      les_houches_example.app:main"
```

Code Listing 3 Example `pyproject.toml` script for the Les Houches parametrisation model.

Note that here the executable `les_houches_exe` is introduced, which is an executable that is specific to this model, and will be used to initialise a fit.

`app.py`

The `app.py` module defines the core application class for the Les Houches model:

```
1
2  """
3  les_houches_example.app.py
4
5  """
6
7  from colibri.app import colibriApp
8  from les_houches_example.config
      import LesHouchesConfig
9
10
11 lh_pdf_providers = [
12     "les_houches_example.model",
13 ]
14
15
16 class LesHouchesApp(colibriApp):
17     config_class = LesHouchesConfig
18
19
20 def main():
21     a = LesHouchesApp(name="
      les_houches", providers=
      lh_pdf_providers)
22     a.main()
23
24
25 if __name__ == "__main__":
26     main()
```

Code Listing 4 Example `app.py` script for the Les Houches parametrisation model.

The `LesHouchesApp` class enables the Les Houches model to function as a reportengine App[76]. This integration provides a structured framework for data processing and report generation.

Some key features are:

- **Provider System:** the `LesHouchesApp` accepts a list of providers (`lh_pdf_providers`) containing modules that are recognized by the application framework.
- **Inheritance Hierarchy:** the `LesHouchesApp` is a subclass of `colibriApp`, which means it automatically inherits all providers from both `Colibri` and validphys, giving access to their full functionality without the need for additional configuration.

`config.py`

The `config.py` script defines the configuration layer for the Les Houches model. It extends `Colibri`'s configuration system to provide a custom model builder and environment.

```python

"""
les_houches_example.config.py

"""

import dill
import logging
from les_houches_example.model import LesHouchesPDF

from colibri.config import Environment, colibriConfig


log = logging.getLogger(__name__)


class LesHouchesEnvironment(Environment):
    pass


class LesHouchesConfig(colibriConfig):
    """
    LesHouchesConfig class Inherits from colibri.config.colibriConfig
    """

    def produce_pdf_model(self, output_path, dump_model=True):
        """
        Produce the Les Houches model.
        """
        model = LesHouchesPDF()
        # dump model to output_path using dill
        # this is mainly needed by scripts/bayesian_resampler.py
        if dump_model:
            with open(output_path / "pdf_model.pkl", "wb") as file:
                dill.dump(model, file)
        return model
```

Code Listing 5 Example `config.py` script for the Les Houches parametrisation model.

The `produce_pdf_model` method creates an instance of the `LesHouchesPDF` model. Therefore, every model should have this production rule.

If `dump_model` is set to `True`, the method serialises the model using `dill` and writes it to `pdf_model.pkl` in the `output_path`, where `output_path` will be the output directory created when running a `Colibri` fit. `pdf_model.pkl` will be loaded by `scripts/bayesian_resampler.py` for resampling.

If `dump_model` is set to `False`, the serialised model will not be written to the disk.

`model.py`

The `model.py` script defines the Les Houches parametrisation model. It does so by defining the `LesHouchesPDF` class, which is based on the more abstract `PDFModel` class within `Colibri`. As described in Sect. 2.1, it needs to specify a list of parameters to be fitted, and a map from these parameters to PDF values on a specified grid in x for each flavour. In the case of the Les Houches parametrisation the parameter names are defined as follows:

```python

"""
les_houches_example.model.py
"""

import jax.numpy as jnp
import jax.scipy.special as jsp
from colibri.pdf_model import PDFModel


class LesHouchesPDF(PDFModel):
    """
    A PDFModel implementation for the Les Houches parametrisation.
    """

    @property
    def param_names(self):
        """The fitted parameters of the model."""
        return [
            "alpha_gluon",
            "beta_gluon",
            "alpha_up",
            "beta_up",
            "epsilon_up",
```

```
                "gamma_up",
                "alpha_down",
                "beta_down",
                "epsilon_down",
                "gamma_down",
                "norm_sigma",
                "alpha_sigma",
                "beta_sigma",
            ]

    @property
    def n_parameters(self):
        """The number of parameters of the model."""
        return len(self.param_names)
```

Code Listing 6 Example extract of a `model.py` script for the Les Houches parametrisation model, showing how `param_names` is filled.

(Note that it would also be possible to automate the name generation for all parameters, for example by setting them all as `w_i` for i running over all parameters. This approach would be convenient for a parametrisation with many parameters.)

For the map to PDF values, we define a function for each PDF, as described in Eq. 3.1. For example, for the gluon PDF, we have:

```

    def _pdf_gluon(
        self, x, alpha_gluon, beta_gluon, norm_sigma, alpha_sigma, beta_sigma
    ):
        """Computes normalisation factor A_g in terms of free parameters and computes the gluon PDF."""
        A_g = (
            jsp.gamma(alpha_gluon + beta_gluon + 2)
            / (jsp.gamma(alpha_gluon + 1) * jsp.gamma(beta_gluon + 1))
        ) * (
            1
            - norm_sigma
            * (jsp.gamma(alpha_sigma + 1) * jsp.gamma(beta_sigma + 1))
            / jsp.gamma(alpha_sigma + beta_sigma + 2)
        )
        return A_g * x**alpha_gluon * (1 - x) ** beta_gluon
```

Code Listing 7 Example extract of a `model.py` script for the Les Houches parametrisation model, showing how a map to PDF values is defined through functions that describe PDFs for each flavour.

The final building block of the `PDFModel` class is the `grid_values_func` function, which takes the parameters of the model and produces a grid that stores the PDF values for each point in x, where the values of x are taken from the `xgrid`. Here we show how it is defined for the gluon PDF parameters, and skip the others for conciseness:

```

    def grid_values_func(self, xgrid):
        """This function should produce a grid values function, which takes
        in the model parameters, and produces the PDF values on the grid xgrid.
        """

        xgrid = jnp.array(xgrid)

        def pdf_func(params):
            """ """
            alpha_gluon = params[0]
            beta_gluon = params[1]
            [...]        # other parameters
            pdf_grid = []

            # Compute the PDFs for each flavour
            gluon_pdf = self._pdf_gluon(
                xgrid, alpha_gluon, beta_gluon, norm_sigma, alpha_sigma, beta_sigma
            )

            [...]        # sigma_pdf, valence3_pdf, t8_pdf

            # Build the PDF grid
            pdf_grid = jnp.array(
                [
                    jnp.zeros_like(xgrid),  # Photon
                    sigma_pdf,  # \Sigma
                    gluon_pdf,  # g
                    valence_pdf,  # V
                    valence3_pdf,  # V3
                    valence_pdf,  # V8 = V
                    valence_pdf,  # V15 = V
                    valence_pdf,  # V24 = V
                    valence_pdf,  # V35 = V
                    jnp.zeros_like(xgrid),  # T3 = 0
                    t8_pdf,  # T8
                    sigma_pdf,  # T15 = \Sigma
                    sigma_pdf,  # T24 = \Sigma
                    sigma_pdf,  # T35 = \Sigma
                ]
```

```
                )
            return pdf_grid

        return pdf_func
```

The `LesHouchesPDF` class completes the abstract methods of the `PDFModel` class. This allows for the definition of a specific model in a way that can be used in the `Colibri` code. The `LesHouchesPDF` class does the following:

- takes a list of flavours to be fitted (`param_names`),
- defines the PDF for each flavour,
- computes grid values.

Having defined this model, it is used in the production rule `produce_pdf_model`, defined in the `config.py` script, shown above. This allows the model to be seen by the rest of the code, so that it can be used to run a fit and perform closure tests.

Installing a model and running a fit

As mentioned above, each model should have its own `pyproject.toml` script, which defines the Python package configuration for this model. Each model can be installed by running

```
pip install -e .
```

in the model directory, which is the one where `pyproject.toml` should be.

This will set up a model-specific executable, which can be used to run fits. In the case of the Les Houches model, this executable is `les_houches_exe`, and can be run simply as:

```
les_houches_exe my_runcard.
    yml
```

which will produce a directory called `my_runcard` with the results. More information on runcard settings and how to process results can be found in the `Colibri` documentation.

Appendix D: The Les Houches parametrisation

Appendix D.1: Free parameters in the Les Houches parametrisation

We adopt the Les Houches parametrisation taken from ref. [56], where it is assumed that the total sea, $\Sigma = u + \bar{u} + d + \bar{d} + s + \bar{s}$, is constrained to be made 40% by up and anti-up, 40% by down and anti-down, and 20% by strange and anti-strange, which means that we can write:

$$
\begin{aligned}
u + \bar{u} &= 0.4\Sigma,\\
d + \bar{d} &= 0.4\Sigma,\\
s + \bar{s} &= 0.2\Sigma.
\end{aligned}
\tag{D.1}
$$

It is also assumed that there is no difference between $\bar{u}$ and $\bar{d}$, so we are only left with four active flavours, namely g, u_v, d_v and Σ. Furthermore, ϵ_g, γ_g, ϵ_Σ and γ_Σ are all set to zero. We are therefore left with the set of equations:

$$
\begin{aligned}
xf_g(x, Q_0) &= A_g\, x^{\alpha_g}\, (1-x)^{\beta_g}\\
xf_{u_v}(x, Q_0) &= A_{u_v}\, x^{\alpha_{u_v}}\, (1-x)^{\beta_{u_v}}\, (1 + \epsilon_{u_v}\sqrt{x} + \gamma_{u_v} x)\\
xf_{d_v}(x, Q_0) &= A_{d_v}\, x^{\alpha_{d_v}}\, (1-x)^{\beta_{d_v}}\, (1 + \epsilon_{d_v}\sqrt{x} + \gamma_{d_v} x)\\
xf_\Sigma(x, Q_0) &= A_\Sigma\, x^{\alpha_\Sigma}\, (1-x)^{\beta_\Sigma}.
\end{aligned}
\tag{D.2}
$$

This amounts to 16 parameters. Moreover not all parameters are independent. A_g is related to A_Σ by the momentum sum rules:

$$
A_g \int_0^1 x^{\alpha_g}\, (1-x)^{\beta_g} dx + A_\Sigma \int_0^1 x^{\alpha_\Sigma}\, (1-x)^{\beta_\Sigma}\, dx = 1,
\tag{D.3}
$$

and the A_{u_v} and A_{d_v} parameters are determined by the valence sum rules:

$$
\begin{aligned}
A_{u_v} \int x^{\alpha_{u_v}-1}\, (1-x)^{\beta_{u_v}}\, (1 + \epsilon_{u_v}\sqrt{x} + \gamma_{u_v} x) dx = 2\\
A_{d_v} \int x^{\alpha_{d_v}-1}\, (1-x)^{\beta_{d_v}}\, (1 + \epsilon_{d_v}\sqrt{x} + \gamma_{d_v} x) dx = 1,
\end{aligned}
\tag{D.4}
$$

leaving 13 free parameters.[5]

Appendix D.2: Normalisations

We can write the expressions for A_g, A_{u_v} and A_{d_v} explicitly by solving the integral spelled out in the sum rules, Equations D.3 and D.4, which are of the form of Euler beta functions, given by:

$$
\int_0^1 dt\, t^{v-1}(1-t)^{w-1} = \frac{\Gamma(v)\Gamma(w)}{\Gamma(v+w)},
$$

where, for positive integer n, $\Gamma(n)$ is defined as:

$$
\Gamma(n) = (n-1)!.
$$

We find that:

$$
A_g = \frac{\Gamma(\alpha_g + \beta_g + 2)}{\Gamma(\alpha_g + 1)\Gamma(\beta_g + 1)}
$$

[5] In ref. [56], ϵ_{u_v} is fixed to its best-fit value, $\epsilon_{u_v} = -1.56$, in order to avoid instability due to a very high correlation between u_v parameters. They therefore left only 12 parameters free to vary. We decide to leave ϵ_{u_v} free because we don't believe we will encounter this problem.

$$\left[1 - A_\Sigma \frac{\Gamma(\alpha_\Sigma + 1)\Gamma(\beta_\Sigma + 1)}{\Gamma(\alpha_\Sigma + \beta_\Sigma + 2)}\right],$$

$$A_{u_v} = \frac{2}{\Gamma(\beta_{u_v} + 1)} \left[\frac{\Gamma(\alpha_{u_v})}{\Gamma(\alpha_{u_v} + \beta_{u_v} + 1)} + \epsilon_{u_v} \frac{\Gamma(\alpha_{u_v} + 1/2)}{\Gamma(\alpha_{u_v} + \beta_{u_v} + 3/2)} + \gamma_{u_v} \frac{\Gamma(\alpha_{u_v} + 1)}{\Gamma(\alpha_{u_v} + \beta_{u_v} + 2)}\right]^{-1},$$

$$A_{d_v} = \frac{1}{\Gamma(\beta_{d_v} + 1)} \left[\frac{\Gamma(\alpha_{d_v})}{\Gamma(\alpha_{d_v} + \beta_{d_v} + 1)} + \epsilon_{d_v} \frac{\Gamma(\alpha_{d_v} + 1/2)}{\Gamma(\alpha_{d_v} + \beta_{d_v} + 3/2)} + \gamma_{d_v} \frac{\Gamma(\alpha_{d_v} + 1)}{\Gamma(\alpha_{d_v} + \beta_{d_v} + 2)}\right]^{-1},$$

Appendix A.3: The Les Houches parametrisation in the evolution basis

We can then use these expressions to find the elements of the evolution basis explicitly, which is given by:

$$\begin{aligned} \Sigma &= u + \bar{u} + d + \bar{d} + s + \bar{s}, \\ T_3 &= (u + \bar{u}) - (d + \bar{d}), \\ T_8 &= (u + \bar{u} + d + \bar{d}) - 2(s + \bar{s}), \\ V &= (u - \bar{u}) + (d - \bar{d}) + (s - \bar{s}), \\ V_3 &= (u - \bar{u}) - (d - \bar{d}), \\ V_8 &= (u - \bar{u} + d - \bar{d}) - 2(s - \bar{s}). \end{aligned} \tag{D.5}$$

Noting that $u_v = u - \bar{u}$, $d_v = d - \bar{d}$ and that, since there are no valence strange quarks, $s_v = s - \bar{s} = 0$, and applying the assumptions stated above, we find:

$$\begin{aligned} T_3 &= 0.4\Sigma - 0.4\Sigma = 0, \\ T_8 &= 0.4\Sigma + 0.4\Sigma - 2 \cdot (0.2\Sigma) = 0.4\Sigma, \\ T_{15} &= T_{24} = T_{35} = \Sigma, \\ V_8 &= u_v + d_v - 2 \cdot 0 = V, \\ V_{15} &= V_{24} = V_{35} = V. \end{aligned} \tag{D.6}$$

Therefore, we are again left with only four active flavours; Σ, V, V_3 and the gluon. We already have an explicit parametrisation for f_Σ and f_g, as stated in Eq. D.2. We have the ingredients to write analogous expressions for f_V and f_{V_3}, which are given by:

$$\begin{aligned} xf_V &= xf_{u_v} + xf_{d_v} \\ &= A_{u_v} x^{\alpha_{u_v}} (1-x)^{\beta_{u_v}} (1 + \epsilon_{u_v}\sqrt{x} + \gamma_{u_v} x) \\ &\quad + A_{d_v} x^{\alpha_{d_v}} (1-x)^{\beta_{d_v}} (1 + \epsilon_{d_v}\sqrt{x} + \gamma_{d_v} x) \end{aligned} \tag{D.7}$$

$$\begin{aligned} xf_{V_3} &= xf_{u_v} - xf_{d_v} \\ &= A_{u_v} x^{\alpha_{u_v}} (1-x)^{\beta_{u_v}} (1 + \epsilon_{u_v}\sqrt{x} + \gamma_{u_v} x) \\ &\quad - A_{d_v} x^{\alpha_{d_v}} (1-x)^{\beta_{d_v}} (1 + \epsilon_{d_v}\sqrt{x} + \gamma_{d_v} x) \end{aligned} \tag{D.8}$$

References

1. L. Del Debbio, T. Giani, M. Wilson, Eur. Phys. J. C **82**(4), 330 (2022). https://doi.org/10.1140/epjc/s10052-022-10297-x
2. S. Amoroso et al., Acta Phys. Polon. B **53**(12), 12 (2022). https://doi.org/10.5506/APhysPolB.53.12-A1
3. M. Ubiali, (2024). arXiv:2404.08508
4. A. Chiefa, M.N. Costantini, J. Cruz-Martinez, E.R. Nocera, T.R. Rabemananjara, J. Rojo, T. Sharma, R. Stegeman, M. Ubiali, JHEP **07**, 067 (2025). https://doi.org/10.1007/JHEP07(2025)067
5. T.J. Hou et al., Phys. Rev. D **103**(1), 014013 (2021). https://doi.org/10.1103/PhysRevD.103.014013
6. S. Bailey, T. Cridge, L.A. Harland-Lang, A.D. Martin, R.S. Thorne, Eur. Phys. J. C **81**(4), 341 (2021). https://doi.org/10.1140/epjc/s10052-021-09057-0
7. S. Alekhin, J. Blümlein, S. Moch, R. Placakyte, Phys. Rev. D **96**(1), 014011 (2017). https://doi.org/10.1103/PhysRevD.96.014011
8. G. Aad et al., Eur. Phys. J. C **82**(5), 438 (2022). https://doi.org/10.1140/epjc/s10052-022-10217-z
9. A.D. Martin, W.J. Stirling, R.S. Thorne, G. Watt, Eur. Phys. J. C **63**, 189 (2009). https://doi.org/10.1140/epjc/s10052-009-1072-5
10. J. Pumplin, Phys. Rev. D **81**, 074010 (2010). https://doi.org/10.1103/PhysRevD.81.074010
11. J. Pumplin, Phys. Rev. D **82**, 114020 (2010). https://doi.org/10.1103/PhysRevD.82.114020
12. L.A. Harland-Lang, T. Cridge, R.S. Thorne, Eur. Phys. J. C **85**(3), 316 (2025). https://doi.org/10.1140/epjc/s10052-025-13934-3
13. A. Barontini, M.N. Costantini, G. De Crescenzo, S. Forte, M. Ubiali, (2025). arXiv:2503.17447
14. R.D. Ball et al., Eur. Phys. J. C **84**(1), 517 (2024). https://doi.org/10.1140/epjc/s10052-024-12772-z
15. Z. Kassabov, M. Ubiali, C. Voisey, JHEP **03**, 148 (2023). https://doi.org/10.1007/JHEP03(2023)148
16. R.D. Ball, R.L. Pearson, Eur. Phys. J. C **81**(9), 830 (2021). https://doi.org/10.1140/epjc/s10052-021-09602-x
17. R. Abdul Khalek et al., Eur. Phys. J. C **79**, 838 (2019). https://doi.org/10.1140/epjc/s10052-019-7364-5
18. R. Abdul Khalek et al., Eur. Phys. J. C **79**(11), 931 (2019). https://doi.org/10.1140/epjc/s10052-019-7401-4
19. R.D. Ball, E.R. Nocera, R.L. Pearson, Eur. Phys. J. C **79**(3), 282 (2019). https://doi.org/10.1140/epjc/s10052-019-6793-5
20. M.N. Costantini, M. Madigan, L. Mantani, J.M. Moore, JHEP **12**, 064 (2024). https://doi.org/10.1007/JHEP12(2024)064
21. R.D. Ball et al., Eur. Phys. J. C **82**(5), 428 (2022). https://doi.org/10.1140/epjc/s10052-022-10328-7
22. R.D. Ball et al., Eur. Phys. J. C (2024). https://doi.org/10.1140/epjc/s10052-024-12731-8
23. E. Moffat, W. Melnitchouk, T.C. Rogers, N. Sato, Phys. Rev. D **104**(1), 016015 (2021). https://doi.org/10.1103/PhysRevD.104.016015
24. G. Watt, R.S. Thorne, JHEP **1208**, 052 (2012). https://doi.org/10.1007/JHEP08(2012)052
25. J. Gao, P. Nadolsky, JHEP **1407**, 035 (2014). https://doi.org/10.1007/JHEP07(2014)035
26. S. Carrazza, S. Forte, Z. Kassabov, J.I. Latorre, J. Rojo, Eur. Phys. J. C **75**(8), 369 (2015). https://doi.org/10.1140/epjc/s10052-015-3590-7

27. S. Carrazza, J.I. Latorre, J. Rojo, G. Watt, Eur. Phys. J. C **75**, 474 (2015). https://doi.org/10.1140/epjc/s10052-015-3703-3
28. S. Carrazza, S. Forte, Z. Kassabov, J. Rojo, Eur. Phys. J. C **76**(4), 205 (2016). https://doi.org/10.1140/epjc/s10052-016-4042-8
29. S. Carrazza, A compression tool for Monte Carlo PDF sets. https://github.com/scarrazza/compressor. Accessed 27 Dec 2025
30. J. Butterworth et al., J. Phys. G **43**, 023001 (2016). https://doi.org/10.1088/0954-3899/43/2/023001
31. R.D. Ball et al., J. Phys. G **49**(8), 080501 (2022). https://doi.org/10.1088/1361-6471/ac7216
32. T. Cridge et al., J. Phys. G **52**, 6 (2025). https://doi.org/10.1088/1361-6471/adde78
33. M.N. Costantini, L. Mantani, J.M. Moore, M. Ubiali, (2025). arXiv:2507.16913
34. A. Candido, L. Del Debbio, T. Giani, G. Petrillo, Eur. Phys. J. C **84**(7), 716 (2024). https://doi.org/10.1140/epjc/s10052-024-13100-1
35. F. Capel, R. Aggarwal, M. Botje, A. Caldwell, O. Schulz, A. Verbytskyi, Phys. Rev. D **110**(1), 014024 (2024). https://doi.org/10.1103/PhysRevD.110.014024
36. J. Albert, C. Balazs, A. Fowlie, W. Handley, N. Hunt-Smith, R.R. de Austri, M. White, Comput. Phys. Commun. **315**, 109756 (2025). https://doi.org/10.1016/j.cpc.2025.109756
37. R. Aggarwal, M. Botje, A. Caldwell, F. Capel, O. Schulz, Phys. Rev. Lett. **130**(14), 141901 (2023). https://doi.org/10.1103/PhysRevLett.130.141901
38. Y.G. Gbedo, M. Mangin-Brinet, Phys. Rev. D **96**(1), 014015 (2017). https://doi.org/10.1103/PhysRevD.96.014015
39. H. Abdolmaleki et al., (2022). arXiv:2206.12465
40. V. Bertone et al., PoS **DIS2017**, 203 (2018). https://doi.org/10.22323/1.297.0203
41. R.D. Ball et al., Nnpdf/nnpdf: nnpdf v4.0.3. (2021). https://doi.org/10.5281/zenodo.5362229
42. R.D. Ball et al., Eur. Phys. J. C **81**(10), 958 (2021). https://doi.org/10.1140/epjc/s10052-021-09747-9
43. J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang. JAX: composable transformations of Python+NumPy programs (2018). http://github.com/jax-ml/jax. Accessed 27 Dec 2025
44. A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page et al., Eur. Phys. J. C **75**, 132 (2015). https://doi.org/10.1140/epjc/s10052-015-3318-8
45. LHAPDF, LHAPDF Documentation. http://projects.hepforge.org/lhapdf/. Accessed 27 Dec 2025
46. R.D. Ball et al., Nucl. Phys. B **809**, 1 (2009). https://doi.org/10.1016/j.nuclphysb.2008.09.037
47. Z. Kassabov, E.R. Nocera, M. Wilson, Eur. Phys. J. C **82**(10), 956 (2022). https://doi.org/10.1140/epjc/s10052-022-10932-7
48. R.D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J.I. Latorre, J. Rojo, M. Ubiali, JHEP **05**, 075 (2010). https://doi.org/10.1007/JHEP05(2010)075
49. A. Candido, S. Forte, F. Hekhorn, JHEP **11**, 129 (2020). https://doi.org/10.1007/JHEP11(2020)129
50. A. Candido, S. Forte, T. Giani, F. Hekhorn, Eur. Phys. J. C **84**(3), 335 (2024). https://doi.org/10.1140/epjc/s10052-024-12681-1
51. A. Candido, F. Hekhorn, G. Magni, Eur. Phys. J. C **82**(10), 976 (2022). https://doi.org/10.1140/epjc/s10052-022-10878-w
52. DeepMind, I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, A. Dedieu, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, G. Papamakarios, J. Quan, R. Ring, F. Ruiz, A. Sanchez, L. Sartran, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, M. Stanojević, W. Stokowiec, L. Wang, G. Zhou, F. Viola, The DeepMind JAX Ecosystem (2020). http://github.com/google-deepmind. Accessed 27 Dec 2025
53. J. Pumplin et al., Phys. Rev. D **65**, 014013 (2001). https://doi.org/10.1103/PhysRevD.65.014013
54. H.L. Lai et al., Phys. Rev. D **82**, 074024 (2010). https://doi.org/10.1103/PhysRevD.82.074024
55. J. Buchner, J. Open Source Softw. **6**(60), 3001 (2021). https://doi.org/10.21105/joss.03001
56. S.I. Alekhin et al., in *HERA and the LHC: A Workshop on the Implications of HERA for LHC Physics: CERN-DESY Workshop 2004/2005 (Midterm Meeting, CERN, 11–13 October 2004; Final Meeting, DESY, 17–21 January 2005)* (CERN, Geneva, 2005), pp. 119–159
57. R.D. Ball et al., JHEP **04**, 040 (2015). https://doi.org/10.1007/JHEP04(2015)040
58. A. Courtoy, J. Huston, P. Nadolsky, K. Xie, M. Yan, C.P. Yuan, Phys. Rev. D **107**(3), 034008 (2023). https://doi.org/10.1103/PhysRevD.107.034008
59. L. Kotz, A. Courtoy, T.J. Hobbs, P. Nadolsky, F. Olness, M. Ponce-Chavez, V. Purohit, (2025). arXiv:2507.22969
60. S. Forte, Z. Kassabov, Eur. Phys. J. C **80**(3), 182 (2020). https://doi.org/10.1140/epjc/s10052-020-7748-6
61. A.M. Cooper-Sarkar, M. Czakon, M.A. Lim, A. Mitov, A.S. Papanastasiou, (2020). arXiv:2010.04171
62. T. Cridge, M.A. Lim, Eur. Phys. J. C **83**(9), 805 (2023). https://doi.org/10.1140/epjc/s10052-023-11961-6
63. S. Alekhin, M.V. Garzelli, S.O. Moch, O. Zenaiev, Eur. Phys. J. C **85**(2), 162 (2025). https://doi.org/10.1140/epjc/s10052-025-13832-8
64. T. Cridge, G. Marinelli, F.J. Tackmann, (2025). arXiv:2506.13874
65. R.D. Ball, A. Barontini, J. Cruz-Martinez, S. Forte, F. Hekhorn, E.R. Nocera, J. Rojo, R. Stegeman, Eur. Phys. J. C **85**(9), 1001 (2025). https://doi.org/10.1140/epjc/s10052-025-14676-y
66. S. Carrazza, C. Degrande, S. Iranipour, J. Rojo, M. Ubiali, Phys. Rev. Lett. **123**(13), 132001 (2019). https://doi.org/10.1103/PhysRevLett.123.132001
67. A. Greljo, S. Iranipour, Z. Kassabov, M. Madigan, J. Moore, J. Rojo, M. Ubiali, C. Voisey, JHEP **07**, 122 (2021). https://doi.org/10.1007/JHEP07(2021)122
68. M. Madigan, J. Moore, PoS **EPS–HEP2021**, 424 (2022). https://doi.org/10.22323/1.398.0424
69. J. Gao, M. Gao, T.J. Hobbs, D. Liu, X. Shen, JHEP **05**, 003 (2023). https://doi.org/10.1007/JHEP05(2023)003
70. Z. Kassabov, M. Madigan, L. Mantani, J. Moore, M. Morales Alvarado, J. Rojo, M. Ubiali, JHEP **05**, 205 (2023). https://doi.org/10.1007/JHEP05(2023)205
71. E. Hammou, Z. Kassabov, M. Madigan, M.L. Mangano, L. Mantani, J. Moore, M.M. Alvarado, M. Ubiali, JHEP **11**, 090 (2023). https://doi.org/10.1007/JHEP11(2023)090
72. M.N. Costantini, E. Hammou, Z. Kassabov, M. Madigan, L. Mantani, M. Morales Alvarado, J.M. Moore, M. Ubiali, Eur. Phys. J. C **84**(8), 805 (2024). https://doi.org/10.1140/epjc/s10052-024-13079-9
73. E. Hammou, M. Ubiali, Phys. Rev. D **111**(9), 095028 (2025). https://doi.org/10.1103/PhysRevD.111.095028
74. X. Shen, S. Amoroso, J. Gao, K. Lipka, O. Zenaiev, Eur. Phys. J. C **84**(11), 1235 (2024). https://doi.org/10.1140/epjc/s10052-024-13585-w
75. S. Eustace, The Poetry contributors. Poetry: Python packaging and dependency management made easy. https://github.com/python-poetry/poetry. Accessed 27 Dec 2025
76. Z. Kassabov, Reportengine: a framework for declarative data analysis. (2019). https://doi.org/10.5281/zenodo.2571601. Accessed 27 Dec 2025