

FEYNCalc 10.2: THIS IS WHY WE CAN HAVE NICE THINGS

Vladyslav Shtabovenko

University of Siegen
Center for Particle Physics Siegen (CPPS)

based on [arXiv:2512.19858](#)

Young Scientists Meeting 2026 of the CRC TRR 257
Aachen

22 September 2026



color meets flavor

- 1 FeynCalc 10.2 and FeynHelpers 2
- 2 Parallel evaluation
- 3 Tensor reduction with zero Gram determinants
- 4 Multiple polylogarithms
- 5 Summary and outlook

FeynCalc 10.2 and FeynHelpers 2



Source: ChatGPT

Why do we automate?

- Feynman's diagrammatic approach: cornerstone of modern perturbative calculations
- High luminosity LHC + possible future colliders: Precise theoretical predictions are in high demand
- Algorithms and hardware got better and faster: many tough calculations can now run on a laptop.
- **How can we make multiloop calculations more accessible and inclusive to nonexperts?**



- **FEYNCALC**: symbolic manipulation of amplitudes, loop integrals and topologies in **MATHEMATICA**.
- **FEYNHELPERS 2** connects this workflow to diagram generators, IBP solvers and numerical integration tools.
- Two practical obstacles: large lists of expressions take too long; special kinematics can break tensor reduction.
- 📦 Today: two features of **FEYNCALC** 10.2 [[Shtabovenko 2025](#)] that help, plus preview of new routines for multiple polylogs.

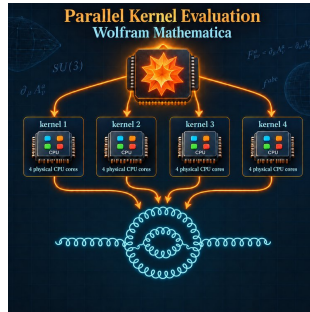


Tools accessible via FEYNHELPERS 2

Tool	Task	Examples
QGRAF [Nogueira, 1993]	Diagram generation	QGCreateAmp
GRAPHVIZ [graphviz.org] / TIKZ-FEYNMAN [Ellis, 2017]	Diagram visualization	(via QGRAF interface)
FIRE [Smirnov & Chukharev, 2020; Smirnov & Zeng, 2024, 2025] + LITERED [Lee, 2012, 2014]	IBP reduction	FIRERunReduction
KIRA [Maierhöfer et al., 2018; Klappert et al., 2021; Lange et al., 2026]	IBP reduction	KiraRunReduction
PYSECDEC [Borowka et al., 2018, 2019; Heinrich et al., 2022, 2024]	Numerical master integrals	PSDCreatePythonScripts
FIESTA [Smirnov, 2016; Smirnov et al., 2022]	Numerical master integrals	FSARunIntegration
LOOPTOOLS [Hahn & Perez-Victoria, 1999]	Numerical Passarino–Veltman functions	LToolsEvaluate
FERMAT [Lewis]	Fast symbolic linear-equation solver	FerSolve
PACKAGE-X [Patel, 2015, 2017]	Analytic 1-loop masters	PaXEvaluate

- 🔴 **PYSECDEC/FIESTA** and **FIRE/KIRA** are independent implementations of the same task
- 🔴 FerSolve speeds up Tdec

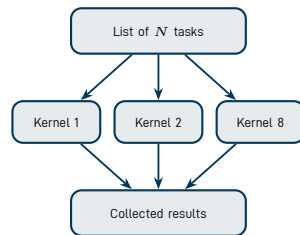
Parallel evaluation



Source: ChatGPT

Many expressions, many kernels

- With modern CPUs parallel evaluation is a must, not a luxury
- MATHEMATICA** has supported parallel kernels since 2008, but making it work efficiently is tricky
 - Simplify**, **Factor**, **Expand** etc. do not automatically split one large expression across kernels.
 - Naively using `ParallelMap` or `ParallelTable` often makes things slower, not faster
- FEYN CALC 10.2** finally supports seamless parallelization over a large range of functions: `Contract`, `DiracSimplify`, `SUNSimplify`, `Collect2`, `FCLoopFindTopologyMappings`, `FCLoopFindIntegralMappings`, ...
- The input must be a `List` (of amplitudes, integrals etc.)
- Evaluation of single expressions (not lists) currently available only for `Collect2`



Two switches: initialize once, enable per call

- Start on a fresh kernel, load **FEYNCALC**, launch the kernels you need and enable parallelization

```
In[1]:= << FeynCalc'
        LaunchKernels[8];
        $ParallelizeFeynCalc = True;
```

- Each kernel has its copy of **FEYNCALC** loaded, kinematics and noncommutative variables automatically shared
- Request parallel execution for each supported operation:

```
In[2]:= result = DiracSimplify[amplitudes, FCPParallelize -> True];
```

- As of now, most routines default to `FCParallelize -> False`.
- The speed-up is significant but not mind-blowing
 - Not everything can be parallelized efficiently
 - Single diagrams often have different complexity
 - Things already possible with **FEYNCALC** can be done faster
 - Calculations that are out of reach remain so

Benchmark inputs from real calculations

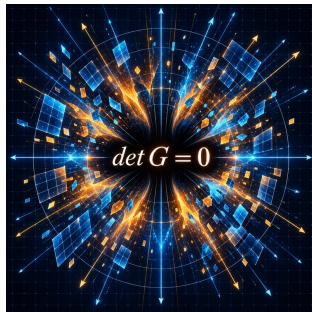
- $\mathcal{T}_{B_c \rightarrow \eta_c}$: 1252 3L topologies (up to 10 props) from a $2 \rightarrow 3$ process calculated for a consistency check for non-relativistic $B_c \rightarrow \eta_c$ form factors at large recoil [Bell et al., 2026].
- $\mathcal{I}_{B_c \rightarrow \eta_c}$: 4000 masters from 1558 topologies of the same calculation.
- $\mathcal{M}_{1L}^{gg \rightarrow HH}$: one-loop $gg \rightarrow HH$ amplitude (massive top, on-shell legs) right after IBP-table insertion; depends on 5 invariants (s, t, u, m_t, m_H), only 4 are independent [Glover & van der Bij, 1988].
- $\mathcal{M}_{2L}^{gg \rightarrow H}$: two-loop $gg \rightarrow H$ amplitude, massive top, on-shell legs [Anastasiou et al., 2007].
- $\mathcal{M}_{2L}^{qq \rightarrow q}$: two-loop QCD qqg -vertex, massive quark plus an auxiliary mass from tadpole-expansion renormalization [Chetyrkin, 2005].

Measured performance

Function	Input	Runtime in s (speed-up vs. 1 kernel)			
		1 kernel	2 kernels	4 kernels	8 kernels
FCLoopFindTopologyMappings	$\mathcal{T}_{B_c \rightarrow \eta_c}$	235.2	139.8 (1.7 $\times$)	71.5 (3.3 $\times$)	49.4 (4.8 $\times$)
FCLoopFindIntegralMappings	$\mathcal{I}_{B_c \rightarrow \eta_c}$	230.8	144.1 (1.6 $\times$)	77.5 (3.0 $\times$)	50.0 (4.6 $\times$)
Collect2 (diagram-wise)	$\mathcal{M}_{1L}^{gg \rightarrow HH}$	137.8	80.6 (1.7 $\times$)	56.8 (2.4 $\times$)	39.3 (3.5 $\times$)
Collect2 (one expression)	$\mathcal{M}_{1L}^{gg \rightarrow HH}$	48.9	37.5 (1.3 $\times$)	30.2 (1.6 $\times$)	25.4 (1.9 $\times$)
DiracSimplify	$\mathcal{M}_{2L}^{gg \rightarrow H}$	55.0	29.0 (1.9 $\times$)	18.5 (3.0 $\times$)	14.7 (3.7 $\times$)
Contract	$\mathcal{M}_{2L}^{gg \rightarrow H}$	15.5	8.4 (1.8 $\times$)	5.0 (3.1 $\times$)	4.1 (3.8 $\times$)
SUNSimplify	$\mathcal{M}_{2L}^{gq \rightarrow q}$	126.6	78.0 (1.6 $\times$)	61.8 (2.0 $\times$)	71.4 (1.8 $\times$)
Contract	$\mathcal{M}_{2L}^{gq \rightarrow q}$	167.7	96.9 (1.7 $\times$)	59.8 (2.8 $\times$)	69.7 (2.4 $\times$)
DiracSimplify	$\mathcal{M}_{2L}^{gq \rightarrow q}$	167.5	99.2 (1.7 $\times$)	68.5 (2.4 $\times$)	75.9 (2.2 $\times$)

- AMD Ryzen 7 PRO 4750U (8 cores), 32 GB RAM, Fedora 43, **MATHEMATICA** 15.0.
- Amdahl's law hitting hard on more than 2 kernels, but still room for optimizations
- Integrals and topology minimizations work best: small chunks of comparable complexity, easy to distribute
- Special case $\mathcal{M}_{2L}^{gq \rightarrow q}$: 36 diagrams, but only 3–4 are really complicated, remaining kernels are idling

Tensor reduction with zero Gram determinants



Source: ChatGPT

Why the usual tensor reduction can fail

For a rank-one 1L integral,

$$I^\mu = \int_k k^\mu f(k, p) = p^\mu A.$$

Contracting with p gives

$$p \cdot I = p^2 A.$$

If $p^2 = 0$, this equation cannot determine A .

Several external momenta: Gram matrix

$$G_{ij} = p_i \cdot p_j.$$

- Standard approach involves $(\det G)^{-1}$.
- $\det G = 0$: the linear system is not solvable.
- Physical meaning (for $n > 1$ in Euclidean space): linearly dependent external momenta.
- Singular kinematics is very common: zero momentum transfer ($g - 2$, EDM), massless form factors, thresholds, ...
- For Passarino–Veltman functions (1L, quadratic propagators) inverse Gram determinant can be avoided [Denner & Dittmaier, 2006; Patel, 2015]
- How to deal with eikonal propagators, multiloop integrals, ...?

Idea: Find a new tensor basis made of linearly independent external momenta. May require auxiliary momenta!

Case 1: two parallel non-light-like external momenta, $\det G(p_1, p_2) = 0$

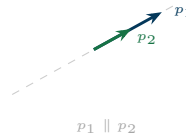
- Construct the tensor basis using p_1 alone.

$$p_2^\mu = c p_1^\mu, \quad p_1^2 \neq 0.$$

- Pay attention to scalar products involving loop momenta:

$$k_i \cdot p_2 = c k_i \cdot p_1.$$

- New basis removes the redundancy without introducing an auxiliary vector.
- Resulting tensor structures are not symmetric in p_1, p_2 .
- Tensor reduction with the new basis works as usual, no extra steps needed



Case 2: a nonzero light-like external momentum, $\det G(p) = 0$

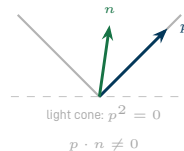
- Need to introduce an auxiliary vector n , with $n^2 = 0$ and $p \cdot n \neq 0$

$$G(p, n) = \begin{pmatrix} 0 & p \cdot n \\ p \cdot n & 0 \end{pmatrix}, \quad \det G(p, n) = -(p \cdot n)^2 \neq 0.$$

- New basis gives

$$I^\mu = p^\mu A + n^\mu B, \quad A = \frac{n \cdot I}{p \cdot n}, \quad B = \frac{p \cdot I}{p \cdot n}.$$

- No loop momenta with open indices left.
- But: scalar integrals involving $k_i \cdot n$ in the numerators.
- The dependence on n cancels out after the IBP reduction.



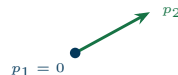
Case 3: a vanishing momentum, $\det G(p_1, p_2) = 0$

- For $p_1^2 = p_1 \cdot p_2 = 0$ with $p_2^2 \neq 0$ we get

$$G(p_1, p_2) = \begin{pmatrix} 0 & 0 \\ 0 & p_2^2 \end{pmatrix}, \quad \det G(p_1, p_2) = 0.$$

- Solving $p_1^\mu = c p_2^\mu$ gives

$$0 = p_1 \cdot p_2 = c p_2^2 \Rightarrow c = 0 \Rightarrow p_1^\mu \equiv 0.$$



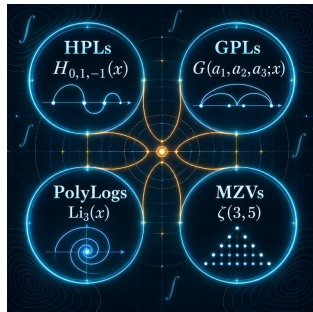
p_1 collapses to the origin

- p_1 is not an independent momentum $\rightarrow$ set $p_1 = 0$ in the integral and reduce with p_2 alone.
- As in Case 1, no auxiliary vector or IBP reduction needed.
- Caveat: light-like p_1 orthogonal to space-like p_2 (e.g. $p_1 = (1, 0, 1, 0)^T$, $p_2 = (0, 1, 0, 1)^T$) leads to the same $\det G \Rightarrow$ Takes us back to Case 2 but with $\{p_1, p_2, n\}$

Treating zero Gram determinants with **FEYNCalc**

- Realistic problems mix all three cases
- Use `FCLoopFindTensorBasis` to construct a new basis automatically
- Give this basis to `TID` or `FCLoopTensorReduce`
- If necessary, do the IBP reduction

Multiple polylogarithms



Source: ChatGPT

What is a multiple polylogarithm?

- Mathematical building blocks behind a large class of loop integrals [Remiddi & Vermaseren, 2000; Goncharov, 2001]
- Goncharov Polylogarithms (GPLs) can be defined as iterated integrals (weight $n = \#$ of integrations):

$$G(a_1, \dots, a_n; z) = \int_0^z \frac{dt}{t - a_1} G(a_2, \dots, a_n; t), \quad G(; z) = 1,$$

- Multiple Polylogarithms (MPLs) are related to GPLs:

$$\text{Li}_{m_1, \dots, m_k}(z_1, \dots, z_k) = \sum_{n_1 > n_2 > \dots > n_k \geq 1} \frac{z_1^{n_1} \dots z_k^{n_k}}{n_1^{m_1} \dots n_k^{m_k}} = (-1)^k G\left(\vec{0}_{m_1-1}, \frac{1}{z_1}, \dots, \vec{0}_{m_k-1}, \frac{1}{z_1 \dots z_k}; 1\right).$$

- Harmonic Polylogarithms (HPLs): GPLs whose alphabet $\vec{a}$ is restricted to $\{0, 1, -1\}$ (but with an extra overall minus for each $a_i = 1$)
- Multiple zeta values (MZVs) are special values at $z = 1$

$$\zeta(m_1, \dots, m_k) = \text{Li}_{m_1, \dots, m_k}(1, \dots, 1), \quad m_1 > 1$$

- Some examples

$$G(0; z) = \ln(z), \quad G(1; z) = \ln(1 - z), \quad G(a; z) = \ln\left(1 - \frac{z}{a}\right), \quad a \neq 0,$$

$$G(\vec{0}_n; z) = \frac{1}{n!} \ln^n z, \quad G(\vec{0}_{n-1}, a; z) = -\text{Li}_n\left(\frac{z}{a}\right), \quad a \neq 0.$$

How to work with MPLs?

- Long-time reliance on community software; no commercial CAS support
- In 2018 **MAPLE** added multiple polylogs [Frellesvig, 2018], but **MATHEMATICA** still required **HPL** [Maitre, 2006, 2012; Huber & Maître, 2006, 2008] and **POLYLOGTOOLS** [Duhr & Dulat, 2019] + **C++ GiNAC** [Bauer et al., 2002]
- Finally, recent **MATHEMATICA** 15 introduced native `HarmonicPolyLog`, `GeneralizedPolyLog`, `MultiplePolyLog`, `MultipleHarmonicNumber` and `MultipleZeta`
- Great, but does not make existing packages obsolete

```
In[6]:= GeneralizedPolyLog[{0, 1, 1}, z]
```

```
Out[6]= -Li3(1 - z) + Li2(1 - z) log(1 - z) +  $\frac{1}{2}$  log(z) log2(1 - z) + ζ(3)
```

```
In[9]:= Integrate[GeneralizedPolyLog[{1, 2, 1, 0}, z], z]
```

```
Out[9]= G(0, 1, 2, 1; z) + G(1, 0, 2, 1; z) + G(1, 2, 0, 1; z) - z (-G(0, 2, 1; z) - G(2, 0, 1; z) + log(z) G(2, 1; z)) +  
z (-G(0, 1, 2, 1; z) - G(1, 0, 2, 1; z) - G(1, 2, 0, 1; z) + log(z) G(1, 2, 1; z)) - log(z) G(1, 2, 1; z) +  
2 (-G(0, 2, 1; z) - G(2, 0, 1; z) + log(z) G(2, 1; z)) - Li2(z) + z (Li2(z) + log(1 - z) log(z)) + z + z (-log(z)) - log(1 - z) log(z)
```

```
In[22]:= Series[GeneralizedPolyLog[{a, 1, 0}, z], {z, 0, 2}]
```

```
Out[22]= (log(z) System`HarmonicPolyLogDump`GPLSeries(GeneralizedPolyLog({a, 1}, 0), {0, 0, 2}) -  
System`HarmonicPolyLogDump`GPLSeries(GeneralizedPolyLog({0, a, 1}, 0), {0, 0, 2}) -
```

New in FEYNCalc (WIP): the FCMpl* toolbox

- We want **FEYNCalc** to be a tool for calculating loop diagrams **and** loop integrals
 - Built-in support of multiple polylogs is essential for that
 - $\text{FCHPL}[\{a_1, a_2, \dots\}, z] \equiv H_{a_1, a_2, \dots}(z)$.
 - $\text{FCGPL}[\{a_1, a_2, \dots\}, z] \equiv G_{a_1, a_2, \dots}(z)$.
 - $\text{FCMZV}[\{a_1, a_2, \dots\}] \equiv \zeta(a_1, a_2, \dots)$.
 - `FCMplConvertHPLToGPL` switches from HPLs to GPLs.
 - `FCMplProductExpand`: shuffle-algebra product expansion.
 - `FCMplRemoveLeadingLetters` / `FCMplRemoveTrailingZeroes`:
shuffle-regularization of leading/trailing letters, e.g. to expose $\ln z$ or $\ln(1 - z)$ singularities.
 - `FCMplSeries`: Taylor expansion of a single GPL/HPL around $z = 0$.
 - `FCMplIntegrate`: integration of GPLs/HPLs.
- 🔍 Basis optimization and argument transformations are still missing (to be added).

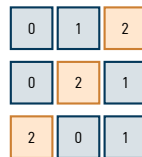
The shuffle algebra

- Words in the letters (indices) form a shuffle algebra
- The product of two GPLs is a sum over all interleavings of their words that preserve each word's internal order.

$$G(\mathbf{a}; z)G(\mathbf{b}; z) = \sum_{\mathbf{c} \in \mathbf{a} \sqcup \mathbf{b}} G(\mathbf{c}; z)$$

- This is a core property of GPLs and HPLs
- Very useful to remove unwanted MPLs from the expression
- Implemented in `FCmplProductExpand`

Example: $w = (0, 1), v = (2)$



blue = w , orange = v ; blue order fixed

$$G(0,1;z)G(2;z) = G(0,1,2;z) + G(0,2,1;z) + G(2,0,1;z)$$

Removing 0-letters: singularities before differentiation

- A trailing 0-letter leads to a logarithmic singularity at $z = 0$ due to $\ln z = G(0; z)$

$$G(1, 0; z) = (1 - \ln z) z + \mathcal{O}(z^2).$$

- Must be removed before computing a Taylor series at $z = 0$, use shuffle algebra

$$G(a_1, \dots, a_k, 0; z) = G(a_1, \dots, a_k; z) G(0; z) - \sum_{\substack{b \in \vec{a} \sqcup 0 \\ b \neq a_1 \dots a_k 0}} G(b; z), \quad a_k \neq 0$$

- Example

$$G(1, 0; z) = G(0; z) G(1; z) - G(0, 1; z).$$

- Then Taylor-expand everything around $z = 0$ except for $G(0; z)^n$
- If there are no trailing zeros, we can directly use the formula [\[Duhr & Dulat, 2019\]](#) with Z -sums [\[Moch et al., 2002\]](#)

$$G(\vec{0}_{m_1-1}, a_1, \dots, \vec{0}_{m_k-1}, a_k; z) = (-1)^k \sum_{n=1}^N \frac{1}{n^{m_1}} \left(\frac{z}{a_1} \right)^n Z_{m_2, \dots, m_k} \left(\frac{a_1}{a_2}, \dots, \frac{a_{k-1}}{a_k}; n-1 \right) + \mathcal{O}(z^{N+1}),$$

- Implemented in **FEYNCALC** via `FCMplRemoveTrailingZeroes` and `FCMplSeries`

The two rules behind the engine

- Differentiation removes the leading letter

$$\frac{d}{dz} G(a_1, a_2, \dots, a_n; z) = \frac{G(a_2, \dots, a_n; z)}{z - a_1}.$$

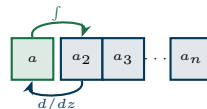
- Integration of a simple pole prepends a letter:

$$\int_{z_0}^{z_1} \frac{dt}{t - a} G(a_2, \dots, a_n; t) = G(a, a_2, \dots, a_n; z_1) - G(a, a_2, \dots, a_n; z_0).$$

- Higher-order poles or polynomial prefactors handled using integration by parts

$$\int_{z_0}^z u'(t) G(a_1, \dots; t) dt = u(t) G(a_1, \dots; t) \Big|_{z_0}^z - \int_{z_0}^z u(t) \frac{G(a_2, \dots; t)}{t - a_1} dt.$$

- Integration available via `FCMplIntegrate`



integration prepends a ;
differentiation removes it

Summary and outlook



Take home messages

- 📦 **FEYNCalc** keeps on getting new cool features
- 📦 Parallel evaluation: do your calculations faster using supported routines.
- 📦 Zero Gram determinants: automatically construct new tensor basis; use auxiliary vectors and IBPs if necessary.
- 🔍 MPLs: the new toolbox for multiple polylogs builds upon **MATHEMATICA** 15's native HPL/GPL/MZV support.