Simplifying Polylogarithms with Machine Learning

Aurélien Dersy
The NSF AI Institute for Artificial Intelligence and Fundamental Interactions
Department of Physics
Harvard University
Cambridge MA 02138
adersy@g.harvard.edu

Matthew D. Schwartz
The NSF AI Institute for Artificial Intelligence and Fundamental Interactions
Department of Physics
Harvard University
Cambridge MA 02138
schwartz@g.harvard.edu

Xiaoyuan Zhang
Department of Physics
Harvard University
Cambridge MA 02138
xiaoyuanzhang@g.harvard.edu

Abstract

In particle physics calculations are centered around Feynman integrals, which are commonly expressed using polylogarithmic functions such as the logarithm or the dilogarithm. Although the resulting expressions usually simplify with an astute application of polylogarithmic identities, it is often difficult to know which identities to apply and in what order. We explore the extent to which machine learning methods are able to help with this creative step. We implement two simplification strategies, one based on an intuitive application of reinforcement learning and one showcasing the potential of language models such as transformers. We demonstrate that the transformer approach is more flexible and holds promise for practical use in symbolic manipulation tasks relevant to mathematical physics.

1 Introduction

In high energy physics Feynman integrals are of primordial interest, being necessary to describe the scattering of elementary particles. In many relevant calculations *polylogarithmic functions* [1, 2, 3, 4, 5, 6] are ubiquitous, with functions such as $\ln(x)$ and $\mathrm{Li}_2(x)$ routinely coming out of the integration [7, 8]. Even if the class of functions to be considered is limited to classical polylogarithms $\mathrm{Li}_n(x)$, simplifying the resulting expression can be challenging [9, 10, 11], as equations can span multiple pages and require judicious applications of mathematical identities. As a practical example, we can consider the following expression which arises in the computation of the Compton scattering total cross section at next-to-leading order (NLO) in quantum electrodynamics [12]:

$$I(r) = \frac{2\pi^2}{9} - \frac{\ln^2 r}{2} + (-\frac{i\pi}{3} + \ln r)\ln(r - z_1) + \ln(rz_1)\ln(r - z_2) \tag{1}$$
$$- \mathrm{Li}_2(1 - z_1) + \mathrm{Li}_2(1 - rz_1) - \mathrm{Li}_2(1 - z_2) + \mathrm{Li}_2(1 - rz_2) \tag{2}$$

where $z_1 = \frac{1+i\sqrt{3}}{2}$ and $z_2 = \frac{1-i\sqrt{3}}{2}$ are cube-roots of unity. Although this expression is correct, it can be simplified further to

$$I(r) = -\frac{1}{3}\mathrm{Li}_2(-r^3) + \mathrm{Li}_2(-r) - \frac{1}{2}\ln^2 r + \frac{\pi^2}{18} \tag{3}$$

through various dilogarithm identities [13]. There is no existing software package (to our knowledge) which can simplify such expressions automatically and one often has to rely on educated guesswork. In this paper we will explore how machine learning methods may address this bottleneck, where we emphasize that our approach will operate on purely symbolic data. For this initial study, we consider classical polylogarithms and functions of a single variable where the problem is already challenging. We first consider *Reinforcement Learning* (RL) to explicitly simplify expressions, prompted by the interesting applications of RL in symbolic domains, notably theorem proving [14, 15, 16, 17], symbolic regression [18] or the untangling of knots [19, 20]. We will then adapt a tool from Natural Language Processing, *transformer networks* [21], for our simplification task following its successes in symbolic integration [22] and solving simple mathematical problems [23, 24].

2 Reinforcement Learning for explicit simplification

Reinforcement Learning setup We first consider a toy problem where our state space corresponds to the set of linear combinations of dilogarithms whose arguments are rational functions over the integers. We restrict ourselves to dilogarithmic expressions which can be reduced using the actions of inversion $\mathrm{Li}_2(x) \to -\mathrm{Li}_2(1/x)$, reflection $\mathrm{Li}_2(x) \to -\mathrm{Li}_2(1-x)$ and duplication $\mathrm{Li}_2(x) \to -\mathrm{Li}_2(-x) + \mathrm{Li}_2(x^2)/2$. Since simplifying single logarithms and constants is not difficult, we drop these terms from our calculus. Symbolic expressions are parsed in prefix notation, following the tree-like structure of mathematical formulas. For instance the mathematical expression $\mathrm{Li}_2(x^2)$ is parsed as ['Li', '2', 'pow', 'x', '2']. Our RL agent will have the option to act on the first dilogarithm term in an expression using either inversion, reflection or duplication. We also allow for a cyclic permutation of the terms as a fourth action, ensuring that every term is reachable. The goal is to produce the shortest expression possible, minimising the number of dilogarithms N^{dilogs}. At a given time t we will reward our agent with $r_t = +1$ if a simplification occurs, so if $N_t^{\text{dilogs}} < N_{t'}^{\text{dilogs}}\ \forall\, t' < t$. We also consider adding a penalty of $\Delta r_t = -0.25$ if the same action (not applicable to permutations) is taken consecutively, without leading to any simplification.

Network architecture All tokens are one-hot encoded and each expression is converted into a graph, based on its tree like structure, before being passed through a *graph neural network* implemented in [25]. We consider the GraphSAGE message passing architecture [26] along with a mean aggregation scheme. Our network uses 2 message passing layers and an embedding dimension of 64. We also experimented with fixed size embeddings for each expression, allowing up to 512 tokens and zero-padding the entries if required. On average this performed slightly worse than the complete graph neural network architecture, which we will retain. As our agent we use the TRPO [27] implementation from [28], with the policy and value networks sharing one layer of size 256, followed by three independent layers of size 128,128 and 64.

Dataset and experiments We let our agents train for 3×10^6 time steps on a single GPU, with the default hyperparameters of [28]. We notably tune the discount factor $\gamma = 0.9$ and the generalized advantage estimate (GAE [29]) $\lambda = 0.9$. For each episode we sample a new expression from a training set with 13,500 examples. Those expressions are generated such that they can all be simplified down to 0 (no dilogarithms) with the help of inversion, reflection or duplication in less than 8 steps. During training an episode is terminated if the simplified form is found or if 50 steps have been taken by the agent. In order to evaluate our agents we let them run on a distinct testing set of 1,500 expressions. The evaluation makes use of a *beam search* of size N. At any given step we consider the top two actions proposed by the agent and keep the N best trajectories in memory. Determining the

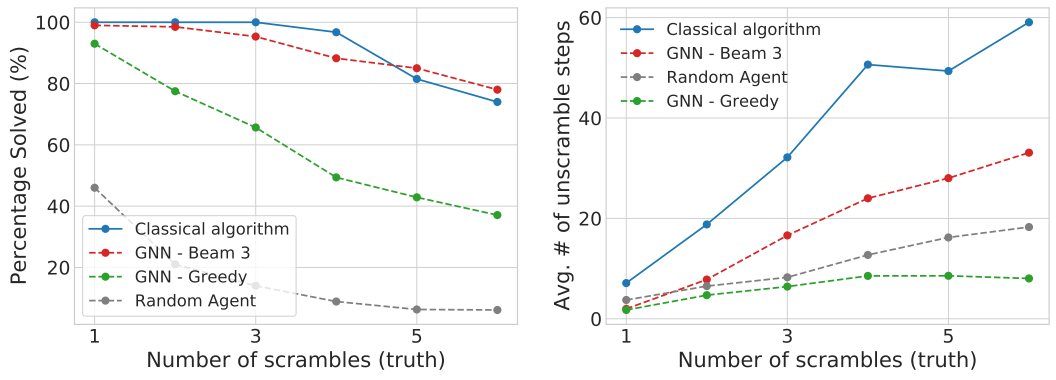

Figure 1: Performance of the best RL agents on the test set of 1,500 dilogarithmic expressions that simplify to 0 (no dilogarithms), as a function of the complexity of the input, given by the number of identities (scrambles) applied to generate the expression. The right panel shows the average number of identities tried (unscrambling steps) by the different agents when solving the examples.

best trajectories is done by making use of the trained value network V and selecting the trajectories that maximize $\tilde{R}_{t+1} = \sum_{k=0}^{t-1} \gamma^k r_k + \gamma^t r_t + V(s_{t+1})$. To benchmark the overall performance we compare to an agent using random actions and to a handcrafted classical algorithm that functions as a modified best first search. This classical algorithm applies all possible actions (on every single term in an expression) and retains the one leading to an immediate simplification. If no simplification is found, a reflection action is performed by default on the first term in order to continue the search. We show on the Fig. 1 the ability of our agents to simplify dilogarithmic expressions along with the number of exploratory steps required.

Challenges and limitations of RL Our trained agents have learned to simplify linear combinations of dilogarithms, reaching an average beam search performance of 89% compared to the 13% of the random agent. However the environment considered was fairly straightforward since the expressions retained were the ones reducing to 0. Training in the general case, where the goal state is not known, is expected to be much harder. Nonetheless RL seems to offer some advantages over a simple classical algorithm. For instance the trained agents reduce the number of evaluations required to find the simplified expression, in a way which does not scale with the total number of available identities and dilogarithm terms. Additional improvements can be considered, namely using the policy and value networks to further guide the search problem (with the A^* search algorithm [30] or a Monte Carlo Tree Search [31]), or implementing a warm start with imitation learning [32].

3 Transformer networks for direct simplification

Transformers for simplification An advantage of the RL approach is its reproducibility: the list of actions taken to arrive at the final answer is known and can be studied and checked. However, demanding reproducibility also imposes a limitation on the way the data is learned and understood. RL algorithms such as ours need to learn an initial embedding layer, require a finite and well defined action space and can suffer from sample inefficiency as commonly seen in other applications of policy gradient methods [33]. If we are only interested in the final simplified form, not the explicit set of identities used to get there, `seq2seq` models are well adapted, with the prime example being the Transformer Network [21]. Our approach is based on the work of [22] (under CC BY-NC 4.0 license), where transformer networks were used to perform a symbolic integration task, showing competitive performance compared to classical algorithms.

To use the transformer we have to convert our task of simplifying equations into a translation one. We explore two different approaches. We first inquire whether it is possible to train a transformer to simplify linear combinations of dilogarithms directly, as in the problem RL was applied to in Section 2. In our second approach we then use the *symbol* (see [9, 34, 35, 36] for a mathematical review) to represent a polylogarithmic expression and deploy a transformer network to look for the simplest expression that is associated with a given symbol.

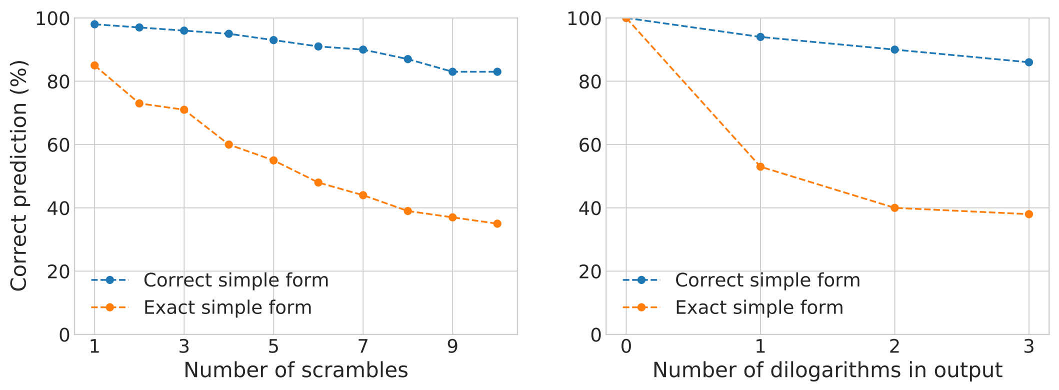

Figure 2: Performance of the transformer in simplifying expressions of dilogarithms as a function of two different measures of complexity. The left panel shows how performance depends on the number of identities (scrambles) applied to generate the expression. The right panel shows performance as a function of the number of terms in the simplified expressions.

Functional simplification For our simplification task we generate a training set of 2.5M simple dilogarithm expressions (with 3 distinct terms or less). We produce associated complicated but equivalent forms by shuffling the simple expressions using inversion, reflection and duplication, using up to 10 identities. We then train a transformer network to generate the simplified form corresponding to a shuffled expression used as input, parsed in a text like prefix notation. We use 3 encoder and decoder layers, an embedding dimension of 512 and train for 8 hours on a single A100 GPU. We then evaluate the performance of our network on a test set of 5,000 expressions and report it on the Fig. 2. The network predicts a single answer which may be exactly the same as the simplified answer in the test set (orange points) or equivalent up to application of inversion and reflection identities which do not increase the complexity, and is therefore still correct (blue points). Predicting the answer gets harder as the length of the expected sequence increases but our model still performs remarkably well, considering that no external calculator is used and that expressions are predicted in symbolic form. Overall we also noticed that performance was robust with respect to changes in the transformer's hyperparameters, offering an advantage over the more fine-tuned RL setup.

Integrating the symbol Another route for simplifying polylogarithms is one that makes use of the *symbol* $\mathcal{S}$, defined in [9]. The symbol extracts the logarithmic differentials from an iterated integral expression and can be easily manipulated with the same product rule satisfied by the logarithm. Evaluating the symbol of an expression and simplifying it is in general more straightforward than dealing with the complete functional form. In practice however recovering the simplest function that corresponds to a given symbol is a hard task, for which no publicly available algorithm exists. For instance while the symbol of the dilogarithm is easy to identify $\mathcal{S}[\mathrm{Li}_2(x)] = -(1-x)\otimes x$, other symbols might correspond to more complicated forms like $\mathcal{S}[\mathrm{Li}_2(x^3) - 3\mathrm{Li}_2(x)] = -3\left(x^2+x+1\right)\otimes x$. This problem of *integrating the symbol* can also be tackled using a transformer network similar to the one described in the last section. Our dataset is composed of about 3.8M polylogarithm expressions $\mathrm{Li}_n(x)$ where the transcendental weight is constrained to $2 \leq n \leq 4$. For each expression we compute its associated symbol in `Mathematica` with `PolyLogTools` [37], using it as the input for the transformer network, from which it has to reconstruct the associated function. We evaluate the performance of our trained network on a test set of 10,000 expressions and compare it to a classical algorithm, which we developed for Li_2 expressions[1]. For higher transcendental weights classical algorithms are not available, nor straightforward to implement. Our results are summarized in the Table. 1 where we can notice that the overall performance remains comparable irrespective of the transcendental weight. In contrast, integrating the symbol by hand is a task that generically becomes considerably harder as the weight increases. Typical failure modes of the network are associated with either long input symbols or complex expressions in the output. This is expected since fewer relevant

[1]Our classical algorithm is developed in `Mathematica` using standard pattern matching tools. All other models and datasets are available under `https://github.com/aureliendersy/ML_Polylogarithms`.

Table 1: Overall solving rate on the test set for the symbol integration task for multiple polylogarithms of different weights. A beam size of N refers to testing the N best guesses of the transformer network.

		Beam Size 1	**Beam Size 5**
Weight 2	Transformer	82%	**91%**
	Classical Algorithm	59%	59%
Weight 3	Transformer	78%	**88%**
Weight 4	Transformer	80%	**89%**

examples would have been encountered during training. Nonetheless the transformer architecture is scalable and we expect that generating more substantial datasets would address that concern.

4 Conclusion

In this paper we have considered whether machine leaning can help with a mathematical problem essential to computations in quantum field theory: polylogarithmic function simplification. We demonstrated that on a simple toy problem RL was a natural machine learning framework to implement, able to efficiently simplify dilogarithmic expressions. However, our training was limited to expressions that could fully simplify to zero using a limited set of identities, such that scaling up would prove challenging. We showed that transformer networks provided a suitable and preferred alternative, capable of guessing the simplified polylogarithmic expression starting from either a complex functional expression, or from a symbol entry. In both cases no general classical algorithm is known and our results suggest that it is conceivable to engineer a suitable machine learning tool for practical usage, for instance by generalizing to arbitrary multivariate functions. Seeing as machine learning has already revolutionized data-driven particle physics (see e.g. [38]), our expectation is that symbolic machine learning methods hold great potential for the future of high energy physics, our work being a first step in that direction.

Acknowledgments and Disclosure of Funding

The computations in this paper were run on the FASRC Cannon cluster supported by the FAS Division of Science Research Computing Group at Harvard University. This research is supported by the National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, `http://iaifi.org/`).

Broader Impact

This paper highlights the use of language models as a tool for the simplification of polylogarithmic functions. As a toy problem it highlights the broader impact that natural language processing could have in the high energy community, acting as a tool for dealing with symbolic manipulations. As opposed to a traditional text generation task, our final output is of mathematical nature and its validity can and should be rigorously assessed. Like any language model the behaviour is highly dependant on the type of data encountered during training. For our purposes we only considered mathematical data which we believe not to pose any ethical challenge.

References

[1] Alexander B. Goncharov. "Multiple polylogarithms, cyclotomy and modular complexes". In: *Math. Res. Lett.* 5 (1998), pp. 497–516. DOI: `10.4310/MRL.1998.v5.n4.a7`. arXiv: `1105.2076 [math.AG]`.

[2] Jonathan M. Borwein et al. "Special values of multiple polylogarithms". In: *Trans. Am. Math. Soc.* 353 (2001), pp. 907–941. DOI: `10.1090/S0002-9947-00-02616-7`. arXiv: `math/9910045`.

[3] A. B. Goncharov. *Multiple polylogarithms and mixed Tate motives*. 2001. arXiv: math/0103059 [math.AG].

[4] A Levin and A Beilinson. “Elliptic polylogarithms”. In: *Proceedings of Symposia in Pure Mathematics*. Vol. 55. Part 2. 1994, pp. 126–196.

[5] Andrey Levin and Georges Racinet. *Towards multiple elliptic polylogarithms*. 2007. DOI: 10.48550/ARXIV.MATH/0703237. URL: https://arxiv.org/abs/math/0703237.

[6] Francis C. S. Brown and Andrey Levin. *Multiple Elliptic Polylogarithms*. 2011. DOI: 10.48550/ARXIV.1110.6917. URL: https://arxiv.org/abs/1110.6917.

[7] A.V. Kotikov. “Differential equations method. New technique for massive Feynman diagram calculation”. In: *Physics Letters B* 254.1 (1991), pp. 158–164. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(91)90413-K. URL: https://www.sciencedirect.com/science/article/pii/037026939190413K.

[8] T. Gehrmann and E. Remiddi. “Two loop master integrals for gamma* —> 3 jets: The Planar topologies”. In: *Nucl. Phys. B* 601 (2001), pp. 248–286. DOI: 10.1016/S0550-3213(01)00057-8. arXiv: hep-ph/0008287.

[9] A. B. Goncharov et al. “Classical Polylogarithms for Amplitudes and Wilson Loops”. In: *Physical Review Letters* 105.15 (Oct. 2010). ISSN: 1079-7114. DOI: 10.1103/physrevlett.105.151605. URL: http://dx.doi.org/10.1103/PhysRevLett.105.151605.

[10] Kai Yan and Xiaoyuan Zhang. “Three-Point Energy Correlator in N=4 Supersymmetric Yang-Mills Theory”. In: *Phys. Rev. Lett.* 129.2 (2022), p. 021602. DOI: 10.1103/PhysRevLett.129.021602. arXiv: 2203.04349 [hep-th].

[11] Tong-Zhi Yang and Xiaoyuan Zhang. “Analytic Computation of three-point energy correlator in QCD”. In: *JHEP* 09 (2022), p. 006. DOI: 10.1007/JHEP09(2022)006. arXiv: 2208.01051 [hep-ph].

[12] Roman N. Lee, Matthew D. Schwartz, and Xiaoyuan Zhang. “Compton Scattering Total Cross Section at Next-to-Leading Order”. In: *Physical Review Letters* 126.21 (May 2021). ISSN: 1079-7114. DOI: 10.1103/physrevlett.126.211801. URL: http://dx.doi.org/10.1103/PhysRevLett.126.211801.

[13] Anatol N. Kirillov. “Dilogarithm Identities”. In: *Progress of Theoretical Physics Supplement* 118 (1995), 61–142. ISSN: 0375-9687. DOI: 10.1143/ptps.118.61. URL: http://dx.doi.org/10.1143/PTPS.118.61.

[14] Cezary Kaliszyk et al. “Reinforcement Learning of Theorem Proving”. In: *CoRR* abs/1805.07563 (2018). arXiv: 1805.07563. URL: http://arxiv.org/abs/1805.07563.

[15] Zsolt Zombori et al. “Towards finding longer proofs”. In: *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer. 2021, pp. 167–186.

[16] Minchao Wu et al. “TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning”. In: *NeurIPS*. 2021.

[17] Gil Lederman et al. “Learning Heuristics for Quantified Boolean Formulas through Reinforcement Learning”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=BJluxREKDB.

[18] Brenden K Petersen et al. “Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=m5Qsh0kBQG.

[19] Sergei Gukov et al. “Learning to Unknot”. In: *Mach. Learn. Sci. Tech.* 2.2 (2021), p. 025035. DOI: 10.1088/2632-2153/abe91f. arXiv: 2010.16263 [math.GT].

[20] Alex Davies et al. “Advancing mathematics by guiding human intuition with AI”. In: *Nature* 600.7887 (2021), pp. 70–74.

[21] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[22] Guillaume Lample and François Charton. *Deep Learning for Symbolic Mathematics*. 2019. arXiv: 1912.01412 [cs.SC].

[23] David Saxton et al. “Analysing Mathematical Reasoning Abilities of Neural Models”. In: *CoRR* abs/1904.01557 (2019). arXiv: 1904.01557. URL: http://arxiv.org/abs/1904.01557.

[24] Aitor Lewkowycz et al. *Solving quantitative reasoning problems with language models*. 2022. arXiv: 2206.14858 [cs.CL].

[25] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[26] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: `1706.02216`. URL: `http://arxiv.org/abs/1706.02216`.

[27] John Schulman et al. *Trust Region Policy Optimization*. 2017. arXiv: `1502.05477 [cs.LG]`.

[28] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[29] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: `10.48550/ARXIV.1506.02438`. URL: `https://arxiv.org/abs/1506.02438`.

[30] Forest Agostinelli et al. "Solving the Rubik's cube with deep reinforcement learning and search". In: *Nature Machine Intelligence* 1.8 (Aug. 2019), pp. 356–363. ISSN: 2522-5839. DOI: `10.1038/s42256-019-0070-z`. URL: `https://doi.org/10.1038/s42256-019-0070-z`.

[31] Stephen McAleer et al. *Solving the Rubik's Cube Without Human Knowledge*. 2018. DOI: `10.48550/ARXIV.1805.07470`. URL: `https://arxiv.org/abs/1805.07470`.

[32] Zhou Liu et al. *Learning to Prove Trigonometric Identities*. 2022. DOI: `10.48550/ARXIV.2207.06679`. URL: `https://arxiv.org/abs/2207.06679`.

[33] Shixiang Gu et al. "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic". In: *Proceedings International Conference on Learning Representations (ICLR)*. Apr. 2017. URL: `https://openreview.net/pdf?id=rkE3y85ee`.

[34] Claude Duhr. "Hopf algebras, coproducts and symbols: an application to Higgs boson amplitudes". In: *Journal of High Energy Physics* 2012.8 (Aug. 2012). ISSN: 1029-8479. DOI: `10.1007/jhep08(2012)043`. URL: `http://dx.doi.org/10.1007/JHEP08(2012)043`.

[35] Claude Duhr. "Function Theory for Multiloop Feynman Integrals". In: *Annual Review of Nuclear and Particle Science* 69.1 (2019), pp. 15–39. DOI: `10.1146/annurev-nucl-101918-023551`. eprint: `https://doi.org/10.1146/annurev-nucl-101918-023551`. URL: `https://doi.org/10.1146/annurev-nucl-101918-023551`.

[36] Rimhak Ree. "Lie elements and an algebra associated with shuffles". In: *Annals of Mathematics* (1958), pp. 210–220.

[37] Claude Duhr and Falko Dulat. "PolyLogTools — polylogs for the masses". In: *Journal of High Energy Physics* 2019.8 (Aug. 2019). ISSN: 1029-8479. DOI: `10.1007/jhep08(2019)135`. URL: `http://dx.doi.org/10.1007/JHEP08(2019)135`.

[38] Matthew D. Schwartz. "Modern Machine Learning and Particle Physics". In: (Mar. 2021). DOI: `10.1162/99608f92.beeb1183`. arXiv: `2103.12226 [hep-ph]`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the abstract, along with the experiments of Section 2 and Section 3
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph of Section 2, Section 3 and the conclusions.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

    (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the footnote in the last paragraph of Section 3. We include a link to our online repository hosting the code and data. The classical algorithm used for weight 2 symbols in Mathematica is proprietary.
    (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the Dataset and experiment paragraph of Section 2. See the Functional simplification paragraph of Section 3.
    (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Variations in results did not impact the overall discussion.
    (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the Dataset and experiment paragraph of Section 2. See the Functional simplification paragraph of Section 3.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
    (a) If your work uses existing assets, did you cite the creators? [Yes] See Ref [22].
    (b) Did you mention the license of the assets? [Yes] See the first paragraph of Section 3 for the code license.
    (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the footnote in the last paragraph of Section 3. We include links to all of our generated datasets.
    (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
    (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
    (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
    (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
    (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]