Scaling Computational Performance of Spherical Harmonics Kernels with Triton

Kin Long Kelvin Lee[1] **Mikhail Galkin**[2] **Santiago Miret**[2]
[1]Data Center and AI Group, Intel Corporation [2]Intel Labs

Abstract

Spherical harmonics are the key ingredient in equivariant neural networks often used in Geometric Deep Learning applications to molecules, proteins, and materials where it is crucial to imbue the model with rotational and translational symmetries. However, computing spherical harmonics at higher orders and on larger inputs often constitutes a major performance bottleneck in equivariant models. In this work, we propose a set of efficient forward and backward kernels implemented in the Triton language for computing spherical harmonics on systems of up to 100 million atoms. Experimentally, the Triton implementation brings improvements of up to $5\times$ in time and up to $3\times$ in memory compared to the popular `e3nn` library while being portable to any GPU accelerator with the Triton backend.

1 Introduction

Equivariant neural networks are among the key components of Geometric Deep Learning [Bronstein et al., 2021] enabling numerous scientific applications from molecular conformation generation [Satorras et al., 2021] and protein-ligand binding [Corso et al., 2023, 2024] to protein folding [Jumper et al., 2021] to materials discovery [Merchant et al., 2023, Miret et al., 2023, 2024]. Equivariant architectures, commonly applied to atomistic systems with 3D coordinates, preserve symmetries and equivariances of geometric quantities under transformations pertaining to the input space [Duval et al., 2023]. For example, the potential energy of a unit cell of a solid-state material does not change after (is *invariant* to) translations, rotations, and reflections (forming the $E(3)$ group) in the 3D space whereas forces applied to atoms change propotionally (are *equivariant*) to rotations (forming the $SO(3)$ group).

Spherical tensors and spherical harmonics, being irreducible representations of the rotation group $SO(3)$, are common components of equvariant architectures such as SE(3)-Transformer [Fuchs et al., 2020], SEGNN [Brandstetter et al., 2021], NequIP [Batzner et al., 2022], MACE [Batatia et al., 2022], eSCN [Passaro and Zitnick, 2023], and Equiformer [Liao and Smidt, 2023, Liao et al., 2024]. At the same time, computing spherical harmonics (especially at higher orders) is bottlenecked by the recurrent nature of the algorithm that becomes more significant on larger systems.

In this work, we propose efficient forward and backward pass kernels implemented in Triton [Tillet et al., 2019] for computing spherical harmonics and enabling higher-order terms on systems of up to 100 million atoms. To the best of our knowledge, this is the first attempt to leverage low-level optimizations and ML compilers for efficient spherical harmonics. Other available implementations [Geiger et al., 2022, Bonev et al., 2023] rely on PyTorch and struggle with time and memory complexity on large inputs and higher orders. Performance-wise, as shown in Section 2.3, the kernels demonstrate up to $5\times$ speedups compared to the popular `e3nn` library [Geiger et al., 2022] used in many equivariant models. On top of that, thanks to the portability of Triton, the kernels can be executed on any GPU regardless of the vendor including Nvidia, AMD and Intel.

AI4Mat-Vienna 2024 Workshop at BOKU University

2 Methodology

2.1 Spherical harmonics

The spherical harmonics (Y_m^l) are a set of polynomial functions that are irreducible representations of the $SO(3)$ symmetry group – by expanding Cartesian features in this basis, we transform them into equivariant ones. The basis is parametrized by l and m, referred to as degree and order, that have physical interpretations representing angular momentum: the former represents the total orbital angular momentum, while the latter is the projection of l along the z axis. For a given degree l there exist $2l + 1$ values of m. For example, as described in Duval et al. [2023], for a $p = (x, y, z)$ point on the unit sphere $S^2 \subset \mathbb{R}^3$, second-order $l = 2$ spherical harmonics $Y_m^{(2)}$ can be expressed as:

$$
\begin{aligned}
&Y_0^{(0)}(p) = c_0 \\
&Y_{-1}^{(1)}(p) = c_1 y, \;\; Y_0^{(1)}(p) = c_1 z, \;\; Y_1^{(1)}(p) = c_1 x, \\
&Y_{-2}^{(2)}(p) = c_2 xy, \;\; Y_{-1}^{(2)}(p) = c_2 yz, \;\; Y_0^{(2)}(p) = \frac{c_2}{2\sqrt{3}}(2z^2 - x^2 - y^2), \\
&\qquad Y_1^{(2)}(p) = c_2 xz, \;\; Y_2^{(2)}(p) = \frac{c_2}{2}(x^2 - y^2),
\end{aligned}
$$

where c_0, c_1, and c_2 are normalization constants.

Currently, equivariant graph neural networks utilize spherical harmonics by mapping cartesian features of n nodes that span $\mathbb{R}^{n\times 3}$, to their spherical harmonic counterparts $\mathbb{R}^{n\times(2l+1)}$, where l is used as a hyperparameter. Subsequent composition of tensor products with learned weights transform these primitive features into non-linear, expressive messages; all the while preserving equivariance. For larger values of l, we nominally obtain a richer feature space: by encoding additional, progressively complex polynomial functions, and by convergence to a complete basis set as $l \to \infty$. The latter, however, has yet to be systematically explored – part in due to the computational demand of evaluating the spherical harmonics, which naively correspond to $O(l^3)$ computational complexity. Although there exist algorithmic ways to reduce $SO(3)$ convolutions to $SO(2)$ convolutions [Passaro and Zitnick, 2023], the polynomial complexity still remains and presents a significant computational challenge. This requirement doubles the computational cost, owing to the fact that their derivatives are required not only for neural network training, but also for evaluating atomic forces (i.e. the derivative of the energy with respect to atom positions).

2.2 Triton & P3 Principles for High-Performance Computing

Triton [Tillet et al., 2019] is a framework for efficient parallel programming with a particular focus on accelerating common building blocks and operations used in deep neural networks. Furthermore, Triton bundles a Python-based language syntax with a compiler. As such, the higher-level language interface is detached from the low-level compilation which makes Triton programs easier to implement and maintain and, at the same time, makes them portable to different hardware that supports the compiler. In high-performance computing (HPC), for example, Triton is among the standard tools for efficient, scalable, and parallelizable training and inference of large neural networks. In particular, Triton kernels implementing fast matrix multiplication, attention, and softmax became widely used with FlashAttention [Dao et al., 2022, Dao, 2024] becoming ubiquitous in Large Language Models (LLMs). Similarly, an efficient implementation of parallel scans enabled scaling state space models [Gu and Dao, 2023] to the sizes of LLMs.

In HPC, performance, portability, and productivity (P3) are tightly coupled concepts that characterize the ability of an application to not only successfully run on different hardware platforms with little to no code changes (or "divergence"), but also able to execute to their respective theoretical limits Pennycook et al. [2016, 2019], Pennycook and Sewall [2021]. While there has yet to be a quantitative P3 study of Triton programs (such as efficient kernels for deep learning workloads), the language shows good promise in terms of P3 by providing a high-level, single source code that does not require hardware specialization by the developer (productivity), whilst offering the ability to perform low-level optimizations such as cache blocking (performance) provided that the platform (portability) has a Triton backend implementation.

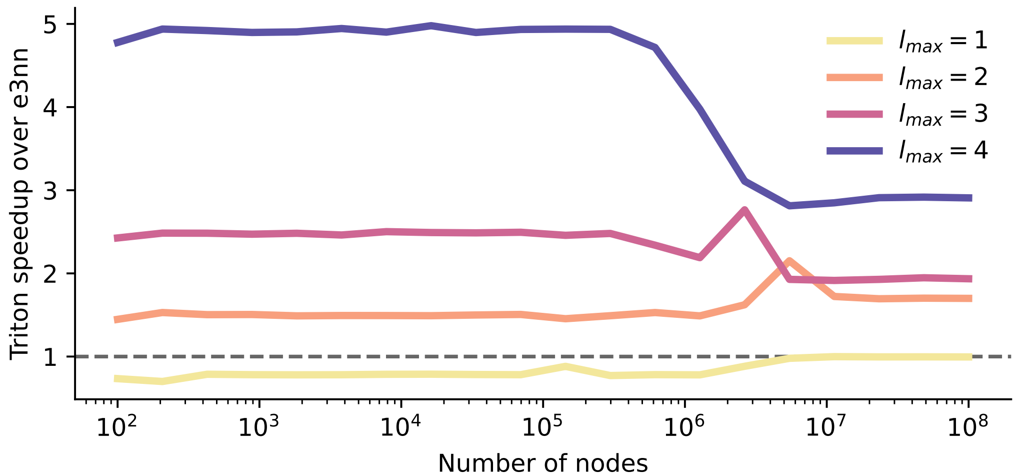

Figure 1: Performance improvement of forward-backward passes using the Triton spherical harmonics kernels relative to `e3nn` for $1 \leq l \leq 4$, as a function of the number of nodes.

2.3 Benchmarking

Experimental setup. Development and benchmarking were performed on Intel® Data Center GPU Max Series 1550 (128 GB HBM2e memory) using oneAPI 2024.0, PyTorch 2.1.0 [Paszke et al., 2019], the XPU version of Intel® Extension for PyTorch 2.1.10, and the 2.1.0 release version of the XPU backend for Triton (commit `69998dd`). The baseline `e3nn` is the latest available pypi release, which is version 0.5.1 from 2022-12-12.

We compare a Triton port of the spherical harmonics functions against those implemented in `e3nn`. Unit tests were written to ensure the two versions produced identical forward and backward results, within a relative tolerance of 10^{-4}. Derivatives up to $l = 4$ are reproduced in the Appendix, and were obtained by symbolic differentiation using `sympy` [Meurer et al., 2017].

Measurements. All results discussed here exclude the time required for just-in-time compilation, which occurs the first time a combination of kernel and input shapes is encountered. For node scaling, we report the median time taken to perform a forward and backward pass based on 90 experiments following 10 warm-up steps (which includes kernel compilation), for a given number of nodes.

3 Results & Discussion

Experimental results on time and memory scaling of Triton kernels are shown in Figure 2 and their comparison with `e3nn` in Figure 1. For training, where a batched graph might comprise upwards of 10^2 to 10^6 atoms, implementing spherical harmonics in Triton afforded $\sim 5\times$ faster forward-backward evaluation than `e3nn`. The gains are even more pronounced on higher-order spherical harmonics ($l = 4$). Furthermore, we observe a constant time footprint (on the order of milliseconds) with small memory consumption, i.e., up to 200 MB for 10^6 atoms.

The improved performance can be attributed to two factors: (1) *cache blocking* and (2) *kernel fusion*. Cache blocking keeps intermediate values that are necessary for computation in cache, thereby mitigating significantly slower read operations from GPU memory. Kernel fusion reduces overhead associated with launching GPU kernels, by virtue of having fewer separate kernels to execute. While `torchscript` offers some degree of fusion, it is unable to perform the same aggressive fusion in the forward and backward passes as a hand-written kernel in Triton.

For $l = 4$, the Triton implementation of spherical harmonics required an average of 39.55/40.96 GB of read/write operations, contrasting against the `torchscript` compiled `e3nn` version, which required

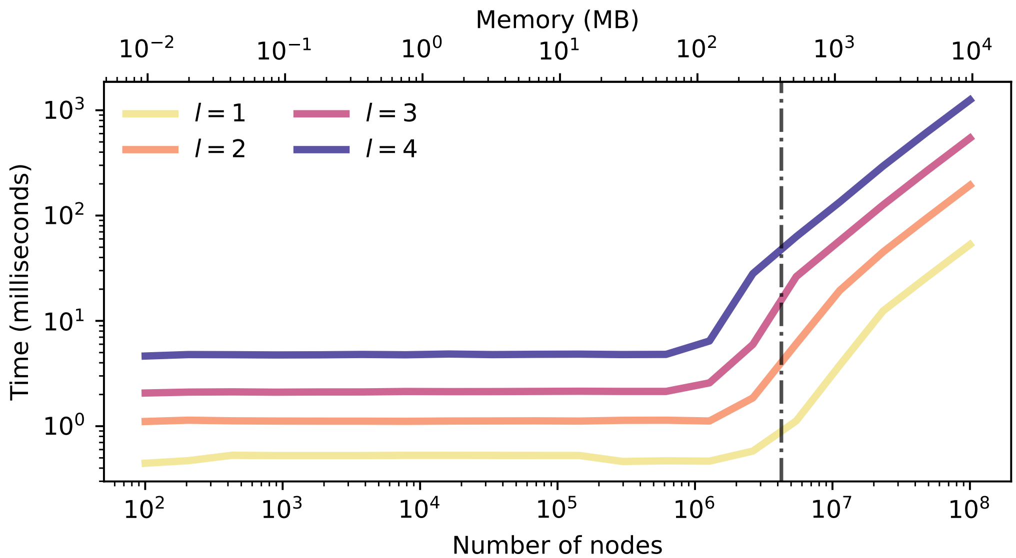

Figure 2: Performance scaling of forward-backward passes with spherical harmonics from `e3nn` as a function of the number of nodes, for $1 \leq l \leq 4$. The top axis corresponds to the expected memory usage for the number of nodes at single floating point precision. The vertical line corresponds to the last-level cache size (408 MB) for the Intel® Data Center GPU Max Series 1550.

104.95/53.97 GB. In other words, the Triton implementation reduces the number of GPU memory reads by a factor of ~2.6.

On larger batches of 10^7 to 10^8 atoms, we still observe up to $3\times$ speedups over `e3nn` albeit time complexity starts to grow reaching the order of a second for 4-th order spherical harmonics on 10^8 atoms. We attribute this decrease in Triton performance to the available last-level cache size on a GPU (408 MB), that is, upon hitting the memory limit, read operations from the main GPU memory start to be prevalent. Nevetheless, the kernels allow to fit 4-th order spherical harmonics of a large 10^8 atomic system within 10 GB of GPU memory which is acceptable for most consumer- and server-grade accelerators.

Finally, we note that Triton is portable across GPUs thanks to the LLVM backend. The same kernels can be reused regardless of vendor/platform as long as they support Triton.

Future Work By providing compute-efficient implementations of spherical harmonics, we hope to spur additional research and development of equivariant neural network architectures, as well as further benchmarking of common architectures which rely on spherical harmonics. The code is open sourced under Apache 2.0, and can be found at https://github.com/IntelLabs/EquiTriton.

Beyond the spherical harmonics, the work we have shown here demonstrates how Triton can be used to improve the performance of scientific AI workflows: tensor products are another essential building block of equivariant models, and the broader community would likely benefit from a similar treatment.

References

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint*, 2021.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Gabriele Corso, Arthur Deng, Nicholas Polizzi, Regina Barzilay, and Tommi Jaakkola. Deep confident steps to new pockets: Strategies for docking generalization. In *International Conference on Learning Representations (ICLR)*, 2024.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

Amil Merchant, Simon Batzner, Samuel S Schoenholz, Muratahan Aykol, Gowoon Cheon, and Ekin Dogus Cubuk. Scaling deep learning for materials discovery. *Nature*, 624(7990):80–85, 2023.

Santiago Miret, Kin Long Kelvin Lee, Carmelo Gonzales, Marcel Nassar, and Matthew Spellings. The open matsci ML toolkit: A flexible framework for machine learning in materials science. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=QBMyDZsPMd.

Santiago Miret, NM Anoop Krishnan, Benjamin Sanchez-Lengeling, Marta Skreta, Vineeth Venugopal, and Jennifer N Wei. Perspective on ai for accelerated materials design at the ai4mat-2023 workshop at neurips 2023. *Digital Discovery*, 2024.

Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Liò, Yoshua Bengio, and Michael Bronstein. A hitchhiker's guide to geometric gnns for 3d atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. SE(3)-transformers: 3D roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33:1970–1981, 2020.

Johannes Brandstetter, Rob Hesselink, Elise van der Pol, E. Bekkers, and M. Welling. Geometric and physical quantities improve E(3) equivariant message passing. *International Conference on Learning Representations*, 2021.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):1–11, 2022.

Ilyes Batatia, Dávid Péter Kovács, Gregor N. C. Simm, Christoph Ortner, and Gábor Csányi. MACE: higher order equivariant message passing neural networks for fast and accurate force fields. In *NeurIPS*, 2022.

Saro Passaro and C Lawrence Zitnick. Reducing SO(3) Convolutions to SO(2) for Efficient Equivariant GNNs. In *International Conference on Machine Learning (ICML)*, 2023.

Yi-Lun Liao and Tess E. Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. In *ICLR*, 2023.

Yi-Lun Liao, Brandon Wood, Abhishek Das*, and Tess Smidt*. EquiformerV2: Improved Equivariant Transformer for Scaling to Higher-Degree Representations. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=mCOBKZmrzD.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

Mario Geiger, Tess Smidt, Alby M., Benjamin Kurt Miller, Wouter Boomsma, Bradley Dice, Kostiantyn Lapchevskyi, Maurice Weiler, Michał Tyszkiewicz, Simon Batzner, Dylan Madisetti, Martin Uhrin, Jes Frellsen, Nuri Jung, Sophia Sanborn, Mingjian Wen, Josh Rackers, Marcel Rød, and Michael Bailey. Euclidean neural networks: e3nn, April 2022. URL https://doi.org/10.5281/zenodo.6459381.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

S. J. Pennycook, J. D. Sewall, and V. W. Lee. A Metric for Performance Portability, November 2016.

S. J. Pennycook, J. D. Sewall, and V. W. Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, 92:947–958, March 2019. ISSN 0167-739X. doi: 10.1016/j.future.2017.08.007.

S. John Pennycook and Jason D. Sewall. Revisiting a Metric for Performance Portability. In *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 1–9, November 2021. doi: 10.1109/P3HPC54578.2021.00004.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

4 Conclusions

A Appendix

A.1 Expressions and derivatives of spherical harmonics

We omit $l = 0$ as it does not contribute gradients. We use use the same notation as `e3nn` to ensure consistency between the kernels regarding the superscript of Y—normally it represents m, but for ease of comparison and comprehension, is used to *index* the polynomial in the same way as implemented in `e3nn` [Geiger et al., 2022]. For the derivatives, we express the collective derivative of a particular degree (e.g. Y_2) with respect to each axis in terms of the gradient vector originating from each order/polynomial (e.g. ∇Y_2^1).

A.1.1 $l = 1$

$$Y_2^0 = \sqrt{3}x \tag{1}$$

$$Y_2^1 = \sqrt{3}y \tag{2}$$

$$Y_2^2 = \sqrt{3}z \tag{3}$$

$$\tag{4}$$

The derivatives are constants $\sqrt{3}$.

A.1.2 $l = 2$

$$Y_2^0 = \sqrt{15}xz \tag{5}$$
$$Y_2^1 = \sqrt{15}xy \tag{6}$$
$$Y_2^2 = \sqrt{15}yz \tag{7}$$
$$Y_2^3 = \sqrt{5}\left(-0.5x^2 + y^2 - 0.5z^2\right) \tag{8}$$
$$Y_2^4 = \frac{\sqrt{15}\left(-x^2 + z^2\right)}{2} \tag{9}$$

The derivatives of Y_2, with respect to each cartesian axis:

$$\frac{\partial Y_2}{\partial x} = \sqrt{15}\nabla Y_2^0 z + \sqrt{15}\nabla Y_2^1 y - \sqrt{5}\nabla Y_2^3 x - \sqrt{15}\nabla Y_2^4 x \tag{10}$$
$$\frac{\partial Y_2}{\partial y} = \sqrt{15}\nabla Y_2^1 x + \sqrt{15}\nabla Y_2^2 z + 2\sqrt{5}\nabla Y_2^3 y \tag{11}$$
$$\frac{\partial Y_2}{\partial z} = \sqrt{15}\nabla Y_2^0 x + \sqrt{15}\nabla Y_2^2 y - \sqrt{5}\nabla Y_2^3 z + \sqrt{15}\nabla Y_2^4 z \tag{12}$$

A.1.3 $l = 3$

$$Y_3^0 = \frac{1}{6}\sqrt{42}Y_2^0 z - Y_2^4 x \tag{13}$$
$$Y_3^1 = \sqrt{7}Y_2^0 y \tag{14}$$
$$Y_3^2 = \frac{1}{8}\sqrt{168}x(4y^2 - x^2 * z^2) \tag{15}$$
$$Y_3^3 = \frac{1}{2}\sqrt{7}y(2y^2 - 3x^2 z^2) \tag{16}$$
$$Y_3^4 = \frac{1}{8}\sqrt{168}z(4y^2 - x^2 * z^2) \tag{17}$$
$$Y_3^5 = \sqrt{7}Y_2^4 y \tag{18}$$
$$Y_3^6 = \frac{1}{6}\sqrt{42}Y_2^4 z - Y_2^0 x \tag{19}$$

The derivatives of Y_3, with respect to each cartesian axis; in this case, the expressions have been partially evaluated by removing dependencies with respect to earlier terms (e.g. ∇Y_2), resulting in literals:

$$\frac{\partial Y_3}{\partial x} = \sqrt{15}\nabla Y_3^0\left(-1.62018517460196x^2 + 1.08012344973464z^2 + 0.540061724867322z^2\right)$$
$$+ 2.64575131106459\sqrt{15}\nabla Y_3^1 yz$$
$$- \nabla Y_3^2\left(4.8605555238059x^2 - 6.48074069840786y^2 + 1.62018517460197z^2\right)$$
$$- 7.93725393319377\nabla Y_3^3 xy - 3.24037034920393\nabla Y_3^4 xz$$
$$- 2.64575131106459\sqrt{15}\nabla Y_3^5 xy - \sqrt{15}\nabla Y_3^6 z\left(2.16024689946929x + 1.08012344973464x\right) \quad (20)$$

$$\frac{\partial Y_3}{\partial y} = 2.64575131106459\sqrt{15}\nabla Y_3^1 xz + 12.9614813968157\nabla Y_3^2 xy$$
$$- \nabla Y_3^3\left(3.96862696659689x^2 - 7.93725393319377y^2 + 3.96862696659689z^2\right)$$
$$+ 12.9614813968157\nabla Y_3^4 yz - 1.3228756555323\sqrt{15}\nabla Y_3^5\left(x^2 - z^2\right) \quad (21)$$

$$\frac{\partial Y_3}{\partial z} = \sqrt{15}\nabla Y_3^0 x\left(1.08012344973464z + 2.16024689946929z\right) + 2.64575131106459\sqrt{15}\nabla Y_3^1 xy$$
$$- 3.24037034920393\nabla Y_3^2 xz - 7.93725393319377\nabla Y_3^3 yz$$
$$- \nabla Y_3^4\left(1.62018517460197x^2 - 6.48074069840786y^2 + 4.8605555238059z^2\right)$$
$$+ 2.64575131106459\sqrt{15}\nabla Y_3^5 yz$$
$$- \sqrt{15}\nabla Y_3^6\left(1.08012344973464x^2 + 0.540061724867322x^2 - 1.62018517460196z^2\right) \quad (22)$$

A.1.4 $l = 4$

$$Y_4^0 = \frac{3}{4}\sqrt{2}(Y_3^0 z + Y_3^6 x) \tag{23}$$

$$Y_4^1 = \frac{3}{4}Y_3^0 y + \frac{3}{8}\sqrt{6}Y_3^1 z + \frac{3}{8}\sqrt{6}Y_3^5 x \tag{24}$$

$$Y_4^2 = -\frac{3}{56}\sqrt{14}Y_3^0 z + \frac{3}{14}\sqrt{21}Y_3^1 y + \frac{3}{56}\sqrt{210}Y_3^2 z + \frac{3}{56}\sqrt{(210)}Y_3^4 x + \frac{3}{56}\sqrt{14}Y_3^6 x \tag{25}$$

$$Y_4^3 = -\frac{3}{56}\sqrt{42}Y_3^1 z + \sqrt{3}28\sqrt{105}Y_3^2 y + \sqrt{3}28\sqrt{70}Y_3^3 x + \frac{3}{56}\sqrt{(42)}Y_3^5 x \tag{26}$$

$$Y_4^4 = -\frac{3}{28}\sqrt{42}Y_3^2 x + \frac{3}{7}\sqrt{7}Y_3^3 y - \frac{3}{28}\sqrt{42}Y_3^4 z \tag{27}$$

$$Y_4^5 = -\frac{3}{56}\sqrt{42}Y_3^1 x + \frac{3}{28}\sqrt{70}Y_3^3 z + \frac{3}{28}\sqrt{105}Y_3^4 y - \frac{3}{56}\sqrt{42}Y_3^5 z \tag{28}$$

$$Y_4^6 = -\frac{3}{56}\sqrt{14}Y_3^0 x - \frac{3}{56}\sqrt{210}Y_3^2 x + \frac{3}{56}\sqrt{210}Y_3^4 z + \frac{3}{14}\sqrt{21}Y_3^5 y - \frac{3}{56}\sqrt{14}Y_3^6 z \tag{29}$$

$$Y_4^7 = -\frac{3}{8}\sqrt{6}Y_3^1 x + \frac{3}{8}\sqrt{6}Y_3^5 z + \frac{3}{4}Y_3^6 y \tag{30}$$

$$Y_4^8 = \frac{3}{4}\sqrt{2}(-Y_3^0 x + Y_3^6 z) \tag{31}$$

The derivatives of Y_4, with respect to each cartesian axis; in this case, the expressions have been partially evaluated by removing dependencies with respect to earlier terms (e.g. $\nabla Y_3, \nabla Y_2$), resulting in literals:

$$\begin{aligned}
\frac{\partial Y_4}{\partial x} = &-\sqrt{15}\nabla Y_4^0(3.43693177121688x^2z + 3.43693177121688x^2z - 1.14564392373896z^3 \\
&- 1.14564392373896z^3) + \sqrt{15}\nabla Y_4^1 y(-4.8605555238059x^2 + 3.24037034920393z^2 \\
&+ 1.62018517460197z^2) - \nabla Y_4^2(0.649519052838329\sqrt{15}x^2z \\
&- 2.77555756156289 \times 10^{-17}\sqrt{15}x^2z + 7.54672942406179x^2z - 2.59807621135332\sqrt{15}y^2z \\
&- 10.0623058987491y^2z + 0.21650635094611\sqrt{15}z^3 + 2.51557647468726z^3) \\
&- \nabla Y_4^3 y(0.918558653543692\sqrt{15}x^2 + 16.0090306546024x^2 - 9.48683298050514y^2 \\
&+ 0.918558653543692\sqrt{15}z^2 + 5.33634355153414z^2 + 0.459279326771846\sqrt{15}\left(x^2 - z^2\right)) \\
&+ \nabla Y_4^4\left(-9.0xy^2 + 2.25xz^2 - 9.0xy^2 + 2.25xz^2 + 4.5x^3\right) \\
&- \nabla Y_4^5 yz\left(1.83711730708738\sqrt{15}x - 0.918558653543692\sqrt{15}x + 10.6726871030683x\right) \\
&- \nabla Y_4^6(2.59807621135332\sqrt{15}xy^2 - 0.21650635094611\sqrt{15}xz^2 + 2.51557647468726xz^2 \\
&+ 10.0623058987491xy^2 - 2.51557647468726xz^2 + 0.21650635094611\sqrt{15}xz^2 - 5.03115294937453x^3 \\
&- 0.433012701892219\sqrt{15}x^3) - \sqrt{15}\nabla Y_4^7 yz\left(6.48074069840786x + 3.24037034920393x\right) \\
&- \sqrt{15}\nabla Y_4^8\left(1.14564392373896xz^2 + 4.58257569495584xz^2 + 1.14564392373896xz^2 - 2.29128784747792x^3\right)
\end{aligned} \tag{32}$$

$$
\begin{aligned}
\frac{\partial Y_4}{\partial y} &= \sqrt{15}\nabla Y_4^1 x\left(-1.62018517460197x^2 + 3.24037034920393z^2 + 1.62018517460197z^2\right) \\
&+ \nabla Y_4^2 xz\left(20.1246117974981y + 5.19615242270663\sqrt{15}y\right) \\
&- \nabla Y_4^3 x(5.33634355153414x^2 - 28.4604989415154y^2 + 0.918558653543692\sqrt{15}z^2 \\
&+ 5.33634355153414z^2 + 0.459279326771846\sqrt{15}\left(x^2 - z^2\right)) \\
&- \nabla Y_4^4\left(9.0x^2y + 9.0x^2y + 9.0yz^2 + 9.0yz^2 - 12.0y^3\right) \\
&- \nabla Y_4^5 z(0.918558653543692\sqrt{15}x^2 + 5.33634355153414x^2 - 28.4604989415154y^2 \\
&+ 5.33634355153414z^2 - 0.459279326771846\sqrt{15}\left(x^2 - z^2\right)) \\
&- \nabla Y_4^6\left(10.0623058987491x^2y - 10.0623058987491yz^2 + 2.59807621135332\sqrt{15}y\left(x^2 - z^2\right)\right) \\
&- \sqrt{15}\nabla Y_4^7 z\left(3.24037034920393x^2 + 1.62018517460197x^2 - 1.62018517460197z^2\right)
\end{aligned} \tag{33}
$$

$$
\begin{aligned}
\frac{\partial Y_4}{\partial z} &= -\sqrt{15}\nabla Y_4^0(1.14564392373896x^3 - 3.43693177121688xz^2 - 3.43693177121688xz^2 \\
&+ 1.14564392373896x^3) + \sqrt{15}\nabla Y_4^1 xy\left(3.24037034920393z + 6.48074069840786z\right) \\
&- \nabla Y_4^2(0.21650635094611\sqrt{15}x^3 - 2.59807621135332\sqrt{15}xy^2 - 10.0623058987491xy^2 \\
&+ 0.649519052838329\sqrt{15}xz^2 - 2.77555756156289\times 10^{-17}\sqrt{15}xz^2 + 7.54672942406179xz^2 \\
&+ 2.51557647468726x^3) \\
&- \nabla Y_4^3 xy\left(-0.918558653543692\sqrt{15}z + 10.6726871030683z + 1.83711730708738\sqrt{15}z\right) \\
&+ \nabla Y_4^4\left(2.25x^2z + 2.25x^2z - 9.0y^2z - 9.0y^2z + 4.5z^3\right) \\
&- \nabla Y_4^5 y(0.918558653543692\sqrt{15}x^2 + 5.33634355153414x^2 - 9.48683298050514y^2 \\
&+ 0.918558653543692\sqrt{15}z^2 + 16.0090306546024z^2 - 0.459279326771846\sqrt{15}\left(x^2 - z^2\right)) \\
&+ \nabla Y_4^6(-0.21650635094611\sqrt{15}x^2z + 2.51557647468726x^2z - 2.51557647468726x^2z \\
&+ 0.21650635094611\sqrt{15}x^2z + 2.59807621135332\sqrt{15}y^2z + 10.0623058987491y^2z \\
&- 5.03115294937453z^3 - 0.433012701892219\sqrt{15}z^3) \\
&- \sqrt{15}\nabla Y_4^7 y\left(3.24037034920393x^2 + 1.62018517460197x^2 - 4.8605555238059z^2\right) \\
&- \sqrt{15}\nabla Y_4^8\left(1.14564392373896x^2z + 4.58257569495584x^2z + 1.14564392373896x^2z - 2.29128784747792z^3\right)
\end{aligned} \tag{34}
$$