Gradient of Clifford Neural Networks

Takashi Maruyama[*]
NEC Laboratories Europe
takashi.maruyama@neclab.eu

Francesco Alesiani[*]
NEC Laboratories Europe
francesco.alesiani@neclab.eu

Abstract

Clifford neural network, geometric neural network over Clifford algebra, is an emerging deep learning model tailored to capture geometrical interactions of physical features. While the models have made promising progress in various tasks, their usage has not yet been explored in scientific tasks that rely on the inverse mode of the networks, which at inference time requires the gradient of the networks over elements of Clifford algebra. In this paper, we give an in-depth theoretical foundation of the gradient of functions over Clifford algebras. To fully utilize the networks' functionality, we extend the notion of the differentiability of general functions between Clifford algebras. We further show that the extended analytic gradient can be numerically derived through widely adopted automatic differentiation modules such as Autograd, which unlocks the full application of the differential modules in any inverse-mode problems on the algebra. We illustrate the promise of using Clifford neural networks in inverse-mode problems in scientific discovery by showing their superior/competitive performance against baselines.

1 Introduction

Clifford neural networks [6, 8, 33, 39, 50, 54, 61], a class of geometric deep learning models [9], have made promising progress in modeling the inherent interactions of physical systems, such as fluid dynamics [6] and multibody interaction systems [50, 8], or geometrical quantities [7]. Typical experiments conducted in the recent literature are categorized as a *forward problem*, where tasks solely rely on the forward-inference mode of neural network models at the inference time. A typical task in this regard is the prediction of the future state given an initial state of a physical system, such as partial differential equations (PDEs) [6, 61] and ordinary differential equations (ODEs) [8, 33, 50, 54]. While solving forward problems is of huge importance, scientific problems and real-world applications include an important class of tasks that require *inverse mode* of models. Therein, models typically require the gradient with respect to their *input* during their inference process. Example applications include inverse-design [1, 60], flow-matching [11, 29], and normalizing flow [41, 52]. We believe that the fundamental reason why many efforts have been put mainly into the forward problem is the fact that the notion of differentiability of Clifford neural networks has not been sufficiently understood. While [14, 24] discuss the differentiability of arbitrary functions between Clifford algebras and derive the analytic gradient for some important classes of functions, some (implicit) condition is imposed on the algebras and the gradient is not well-defined for some very useful class of Clifford algebras.

Contributions. As a first step towards the application of Clifford neural networks to inverse-mode problems, i) we extend the notion of the differentiability of functions between Clifford algebras to Clifford algebras with any signature to enable potential theoretical analysis of the gradients to arbitrary inverse-mode problems over Clifford algebras. To allow for their experimental usage, ii) we further show that the analytic gradient of the functions between Clifford algebras is compatible with the gradient of the functions restricted on their underlying vector spaces, which eventually ensures the use of automatic differentiation modules such as Autograd [44] and numerical derivation of the gradient, equivalent to the analytic gradient, on any types of Clifford algebras; iii) we illustrate the promise of using the neural networks defined over Clifford algebra across a variety of inverse-mode

[*]Equal contribution

problems in scientific discovery. We demonstrate in respective experiments that the gradient of the networks yields superior or competitive performance against strong baseline models. The code for the experiments is provided in `https://github.com/nec-research`.

2 Background

Clifford Algebra. We start by introducing Clifford algebra [38], also known as geometric algebra [25], over a real vector space V of finite dimension n, and some of its key properties. We follow similar notation and definition as in [50, 61]. The Clifford algebra $Cl(V, \mathfrak{q})$ with a quadratic form $\mathfrak{q} : V \to \mathbb{R}$ is a vector space generated by the l-fold tensor product of an arbitrary basis $\{\boldsymbol{e}_i\}_{i=1}^{n}$ of V with an equivalence relation $\mathfrak{q}(\boldsymbol{v}) = \boldsymbol{v} \otimes \boldsymbol{v}$ $(\forall \boldsymbol{v} \in V)$. Then, every element $\boldsymbol{x} \in Cl(V, \mathfrak{q})$ may be written with finite indices $I_m = \{i_1 < \cdots < i_m\} \subset \{1, 2, \cdots, n\}$

$$\boldsymbol{x} = \sum_{m=0}^{n} \sum_{I_m} x_{I_m} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m}, \quad x_{I_m} \in \mathbb{R}. \tag{1}$$

Note that $I_m = \emptyset$ for $m = 0$. The expression $\boldsymbol{v} \otimes_{\mathfrak{q}} \boldsymbol{w}$ of elements $\boldsymbol{v}, \boldsymbol{w} \in V$ represents the *geometric product* of $\boldsymbol{v}, \boldsymbol{w}$, which defines a product on $Cl(V, \mathfrak{q})$ and charactrizes $Cl(V, \mathfrak{q})$ as an algebra. The product of $\boldsymbol{x}, \boldsymbol{y} \in Cl(V, \mathfrak{q})$ runs all the pair of $\boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m}$ composing respective $\boldsymbol{x}$ and $\boldsymbol{y}$, but some of the basis elements $\boldsymbol{e}_i$ is reduced to a scalar because of the relation $\mathfrak{q}(\boldsymbol{e}_i) = \boldsymbol{e}_i \otimes_{\mathfrak{q}} \boldsymbol{e}_i$,

$$(\boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r}) \otimes_{\mathfrak{q}} (\boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s}) = \prod_{u=0}^{t-1} \mathfrak{q}(e_{k_{r+s-u}})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}}). \tag{2}$$

Clifford Neural Networks. Clifford algebra is incorporated into many machine learning models, such as message passing neural networks (MPNNs) [50], simplicial MPNNs [33], multilayer perceptron models [39], convolutional neural networks [61], and transformers [8]. Those models are typically represented as functions composed of polynomials defined over Clifford algebra. The polynomials belong to the algebra $\mathbb{R}[X_1, X_2, \cdots, X_c]$ of polynomials (of any order) in the coefficients of $\mathbb{R}$ with c variables, in which the sum and product are defined as the sum and geometric product of $Cl(\mathbb{R}^n, \mathfrak{q})$. Therefore, the models in the literature are instances of the composition and concatenation of polynomials $\boldsymbol{F} \in \mathbb{R}[X_1, X_2, \cdots, X_c]$, each of which serves as a map from a product space of Clifford algebras (of channel dimension c) to algebra:

$$\underbrace{Cl(\mathbb{R}^n, \mathfrak{q}) \times \cdots \times Cl(\mathbb{R}^n, \mathfrak{q})}_{c} \xrightarrow{\boldsymbol{F}} Cl(\mathbb{R}^n, \mathfrak{q}). \tag{3}$$

We name this class of models as Clifford neural networks.

3 Gradient of functions between Clifford algebras

In this section, we show our theoretical results on the differentiability of functions over Clifford algebra.

Differentiable function on Clifford algebra. Let $\mathfrak{g}_V$ be an Euclidean metric for V, i.e., a symmetric, non-degenerate and positive bilinear form $\mathfrak{g}_V : V \times V \to \mathbb{R}$. The metric induces a metric $\mathfrak{g}_{Cl(V,\mathfrak{q})}$ on $Cl(V, \mathfrak{q})$ of dimension 2^n, which is detailed in Appendix A.1 and A.2. With this induced metric, the $\boldsymbol{a}$-directional gradient of $\boldsymbol{F}$ at $\boldsymbol{x}_0 \in Cl(V, \mathfrak{q})$ in the direction $\boldsymbol{a} \in Cl(V, \mathfrak{q})$ is defined as

$$\boldsymbol{F}'_{\boldsymbol{a}}(\boldsymbol{x}_0) = \lim_{\lambda \to 0} \frac{\boldsymbol{F}(\boldsymbol{x}_0 + \lambda \boldsymbol{a}) - \boldsymbol{F}(\boldsymbol{x}_0)}{\lambda}, \quad \boldsymbol{a} \in Cl(V, \mathfrak{q}). \tag{4}$$

The original definition is given in [14, 24]. Here, the distance of the space, used when taking infinitely small λ, is defined by a norm $||\boldsymbol{x}|| = \sqrt{\mathfrak{g}_{Cl(V,\mathfrak{q})}(\boldsymbol{x}, \boldsymbol{x})}$. We call $\boldsymbol{F}$ *differentiable* when the limit exists for any directional vector $\boldsymbol{a} \in Cl(V, \mathfrak{q})$ and $\boldsymbol{x}_0$ (and its associated gradient is continuous). Another prerequisite for this notion is detailed in Appendix A.

Connection on gradient between base space and associated Clifford algebra. The quadratic form $\mathfrak{q}$, as defined in Section 2, defines a bilinear form $\mathfrak{b}(v, w) = \frac{1}{2}(\mathfrak{q}(\boldsymbol{v} + \boldsymbol{w}) - \mathfrak{q}(\boldsymbol{v}) - \mathfrak{q}(\boldsymbol{w})) : V \times V \to \mathbb{R}$. Throughout the rest of this paper, we assume the bilinear form $\mathfrak{b}$ to be an inner product of signature (p, q, r), i.e., $\mathfrak{b}(\boldsymbol{v}, \boldsymbol{w}) = \mathfrak{b}^{p,q,r}(\boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{v}^{\mathrm{T}} \Delta^{p,q,r} \boldsymbol{w}$ with matrix $\Delta^{p,q,r} = \mathrm{diag}(\underbrace{1, \cdots, 1}_{p}, \underbrace{-1, \cdots, -1}_{q}, \underbrace{0, \cdots, 0}_{r})$, which leads to the following equivalent relations:

$$\mathfrak{q}(\boldsymbol{e}_i) = \begin{cases} +1, & 1 \leq i \leq p, \\ -1, & p + 1 \leq i \leq p + q, \\ 0, & p + q + 1 \leq i \leq p + q + r. \end{cases}$$

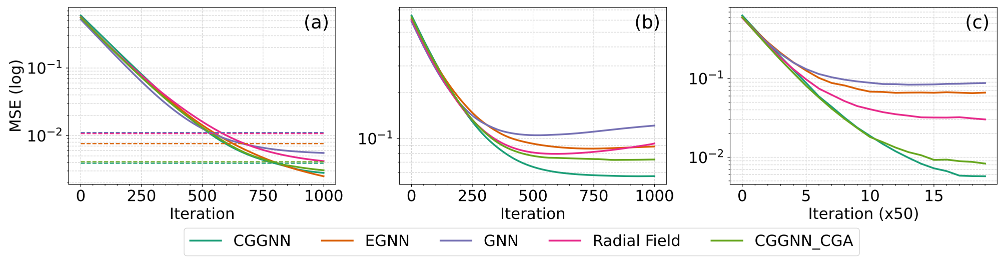

Figure 1: **Numerical results of inverse optimization for input of neural network model** f_θ. (a) shows optimization curve (Eq. 5) for inverse-design at different iteration steps. The horizontal dotted lines are the final test errors observed in the training phase of respective models, which are comparable to the test errors reported in [50]. (b) is the MSE error between ground-truth input velocity and optimized input, and (c) shows MSE error between the ground-truth target state and simulated target with respect to the optimized velocity input.

We also denote $\mathbb{R}[X_1, \cdots, X_c]_{p,q,r}$ as the set of polynomial functions on the Clifford algebra whose geometric product $\otimes_\mathfrak{q}$ is associated with the bilinaer form with signature (p, q, r). We here claim that all functions in $\mathbb{R}[X_1, \cdots, X_c]_{p,q,r}$ are differentiable for **any** signatures (p, q, r), which is an extension of the derivative defined for $(p, q, 0)$ in [14, 24]. By Eq. (3) in Section 2, the claim implies that Clifford neural networks are differentiable. The formal claim is given in Appendix C.

With Proposition C.1, we further elaborate the connection between gradients of functions between Clifford spaces and its base spaces: Suppose that we have the following embedding (inc) and projection (proj) maps between V and $Cl(V, \mathfrak{q})$:

$$\mathrm{inc} : V \hookrightarrow Cl(V, \mathfrak{q}),\ \boldsymbol{v} \mapsto \sum_{k=1}^{n} \mathfrak{g}_V(\boldsymbol{v}, \boldsymbol{e}_k)\boldsymbol{e}_k,\ \ \mathrm{proj} : Cl(V, \mathfrak{q}) \twoheadrightarrow V,\ \sum_{m=0}^{n}\sum_{I_m} v_{I_m}\boldsymbol{e}_{I_m} \mapsto \sum_{I_1} v_{I_1}\boldsymbol{e}_{I_1}.$$

Corollary 3.1. *For* $\forall \boldsymbol{F} \in \mathbb{R}[X]_{p,q,r}$*, its restriction to the base space* V *by* inc *and* proj

$$V \xhookrightarrow{\mathrm{inc}} Cl(V, \mathfrak{q}) \xrightarrow{\boldsymbol{F}} Cl(V, \mathfrak{q}) \xrightarrow{\mathrm{proj}} V$$

is a differentiable function between V *with respect to the metric* g_V*. In particular, when* $V = \mathbb{R}^n$ *and its basis is the standard orthonormal basis, the function* $\mathrm{proj} \circ F \circ \mathrm{inc}$ *is differentiable on* $\mathbb{R}^n$ *with respect to the canonical differentiable structure on Euclidean space.*

This corollary ensures that the gradient of $\mathrm{proj} \circ \boldsymbol{F} \circ \mathrm{inc}$ is the "standard" gradient over the Euclidean spaces, and hence coincides with the gradient obtained through an automatic differentiation module.

4 Experiments

In this section, we illustrate experimentally the promising usage of Clifford neural networks in an inverse-design of physical features and a challenging density modelling experiments in science discovery. We also conduct runtime analysis for the analytic gradient and the gradient obtained through Autograd. The details of the experiments are described in Appendix E, G, and J.

Inverse design. We base our inverse-design experiment on N-body interaction simulation [28, 50, 53]. Our objective in this experiment is to find out optimal initial parameters $\boldsymbol{x}^*$ of the simulation such as coordinates and/or velocities that minimize the distance between their predicted future coordinate and given target coordinate. Formally, given initial parameters $\boldsymbol{x} \in \mathbb{R}^{m \times N}$ with m-features and future target coordinate $\boldsymbol{y} \in \mathbb{R}^{3 \times N}$, we optimize $\boldsymbol{x}$ by minimizing the following objective function

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{f_\theta}(\boldsymbol{x})) = \mathrm{MSE}(\boldsymbol{y}, \boldsymbol{f_\theta}(\boldsymbol{x})). \tag{5}$$

We evaluate the performance of Clifford neural networks with a couple of strong baselines. The details of the baselines are given in Appendix G.1 All the networks take physical features of positions and velocities of the bodies (and some invariant features) as input, and are trained to predict bodies' positions after 1000 timesteps, where we use a similar experimental setup for the models as in [5, 8, 50, 53] (We also present the details in Appendix G.1.)

Table 1: **Comparison of the inverse design results with baselines.** Comprehensive results are in Appendix J.2

	Coordinate		
	Obj	InitDist	TarDist
EGNN	1.31	13.23	20.38
CGGNN	**0.74**	**5.33**	**9.52**

Figure 1 shows the results of the inverse design with Clifford Group-Equivariant GNN (CGGNN) with respect to the velocity input. We observed that CGGNN outperforms or is competitive with other baselines in all metrics (Tab.1). Although in Figure 1 (a) the objective of CGGNN is slightly worse than that of EGNN, in other metrics (b) and (c) CGGNN outperforms all the baselines by a wide margin. This indicates that CGGNN can perform reasonable and robust inverse design, while the other baselines fall into adversarial modes early in the inverse design iteration. We further analyze the robustness of the proposed approach, by varying the noise variance σ for the initial parameters $\hat{\boldsymbol{x}}$, in Appendix J.3. We also report the experimental results of the geometric algebra transformer [8] (with the signature $(3, 0, 1)$) including the other baselines in Appendix J.2.

Sampling from distribution. To further investigate the use of gradients in the Clifford Algebra, we adapt Clifford neural networks to continuous normalizing flow. We consider a Double Well (DW) and Lennard-Jones (LJ) particle systems, as presented in [30], which model the interactions among particles. **DW4** consists of four particles moving in a 2 dimensional space whose energy depends on a pair of particles. **LJ13** consists of 13 particles and models the potential between molecules as *Lennard-Jones* potentials. In Appendix B, we provide the details on how probability density functions are defined on Clifford algebra and how the change of variable is implemented when using Clifford neural networks for the gradient flow equation. Following the experimental setup of [53], we use 10^3 samples for testing and validation, while the training is performed on $10, 10^2$ and 10^3 samples. We compare state-of-the-art E(n)-equivariant flow architectures, whose details are given in Appendix E. Table 2 shows the results of DW4 and LJ13 experiments. We observe that the performance of CNF with CGGNN models is better or comparable to the other baselines. We also compare the performance of CGGNN with that of E-NF with the increased number of hidden-channel dimension, to ensure that both of the models have a comparable number of hidden units for a fair comparison. The performance of CGGNN is still comparable to or better than those baselines. These results indicate that the back-propagation through Clifford neural networks can carry informative Jacobian to transform density functions across time.

Table 2: Comparison of the Negative Log Likelihood on the test partition on DW4 and LJ3 dataset.

	DW4 ($n = 2$)		**LJ13** ($n = 3$)	
# training samples	10^2	10^3	10	10^2
E-NF	$8.31^{\pm 0.05}$	$\mathbf{8.15^{\pm 0.10}}$	$33.12^{\pm 0.85}$	$\mathbf{30.99^{\pm 0.95}}$
E-NF (24×2^n)	$\mathbf{8.24^{\pm 0.06}}$	$8.33^{\pm 0.09}$	$\mathbf{31.33^{\pm 0.30}}$	$\mathbf{30.61^{\pm 0.16}}$
CGGNN (24)	$8.80^{\pm 0.32}$	$8.56^{\pm 0.04}$	$\mathbf{31.36^{\pm 0.55}}$	$\mathbf{30.35^{\pm 0.18}}$

Analytic gradient and Autograd gradient. We also report the computation time when computing the gradient in the analytic form given in [14] and when using the Autograd version as proposed in our work. The aim of the runtime experiment is to clarify the range of parameters with which the computation of Autograd gradients outperforms that of analytical gradients. While we compare the runtimes, we keep comparing the values of the two gradients to make sure that the two gradients are the same. We report their ratio of the time of computing the analytic gradient to the time of computing Autograd gradient. We here consider the gradient of polynomial functions of increasing order x^k for $k = 2$ with different signatures $\{(2, 0, 0), (3, 0, 0), (3, 0, 1), (4, 1, 0)\}$, for different batch sizes 1 and 100 and channel sizes 1 and 10. We choose the polynomial functions since the polynomials are the primary components composing Clifford neural networks, as shown with Eq. (3) in Section 2.

Table 3: **Ratio of the computation time of analytic gradient to that of autograd gradients.**

		(batch size, channel size)			
signature	k	(1, 1)	(1, 10)	(100, 1)	(100, 10)
(2, 0, 0)	2	2.08	2.29	2.04	1.20
(3, 0, 0)	2	4.89	3.45	2.21	1.12
(3, 0, 1)	2	3.75	3.04	3.70	1.11
(4, 1, 0)	2	11.02	8.40	1.84	0.50

In Table 3, we observe that the computation of the analytic gradient is more efficient with larger batchsize, channel sizes, and larger polynomial degree and signature. This observation is a typical tendency whenever comparing the runtime between analytical solution and Autograd's solution, which supports that the gradient computation by Autograd is performed properly. Comprehensive results are reported in Appendix I.

5 Conclusions

In this paper, we show that the gradient of functions between Clifford spaces is compatible with the gradient of functions in the Euclidean spaces. This theoretical connection was previously overlooked. With this theoretical result, we can immediately show that the gradient obtained through Autograd coincides with the analytical gradient. We also provide empirical evidence of the utility of the gradient of Clifford neural networks in the context of inverse-mode tasks. We expect that our exposition inspires future research into better-aligned inverse-mode problems.

References

[1] Kelsey R Allen, Tatiana Lopez-Guavara, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, Jessica B Hamrick, and Tobias Pfaff. Inverse design for fluid-structure interactions using graph network simulators. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[2] Michael Athanasopoulos, Hassan Ugail, and Gabriela González Castro. Parametric design of aircraft geometry using partial differential equations. *Advances in Engineering Software*, 40(7):479–486, 2009.

[3] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J Bekkers, and Max Welling. Geometric and physical quantities improve e(3) equivariant message passing. In *International Conference on Learning Representations*, 2022.

[6] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for PDE modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

[7] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[8] Johann Brehmer, Pim de Haan, Sönke Behrends, and Taco Cohen. Geometric algebra transformer. In *Advances in Neural Information Processing Systems*, volume 37, 2023.

[9] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[10] Keith T Butler, Jarvist M Frost, Jonathan M Skelton, Katrine L Svane, and Aron Walsh. Computational materials design of crystalline solids. *Chemical Society Reviews*, 45(22):6138–6146, 2016.

[11] Ricky T. Q. Chen and Yaron Lipman. Flow matching on general geometries. In *The Twelfth International Conference on Learning Representations*, 2024.

[12] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[13] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

[14] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[15] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[16] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 01 2009.

[18] Virginia V. Fernández, Antonio M. Moya, and Waldyr A. Rodrigues. Euclidean clifford algebra. *Advances in Applied Clifford Algebras*, 11(3):1–21, Oct 2001.

[19] Virginia V. Fernández, Antonio M. Moya, and Waldyr A. Rodrigues. Extensors. *Advances in Applied Clifford Algebras*, 11(3):23–40, Oct 2001.

[20] Virginia V. Fernández, Antonio M. Moya, and Waldyr A. Rodrigues. Multivector functions of a multivector variable. *Advances in Applied Clifford Algebras*, 11(3):79–91, Oct 2001.

[21] Amir Gholaminejad, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 730–736. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[23] David Hestenes. Hamiltonian mechanics with geometric calculus. In Zbigniew Oziewicz, Bernard Jancewicz, and Andrzej Borowiec, editors, *Spinors, Twistors, Clifford Algebras and Quantum Deformations*, pages 203–214, Dordrecht, 1993. Springer Netherlands.

[24] David Hestenes and Garret Sobczyk. *Clifford algebra to geometric calculus : a unified language for mathematics and physics*. D. Reidel ; Distributed in the U.S.A. and Canada by Kluwer Academic Publishers, Dordrecht; Boston; Hingham, MA, U.S.A., 1984.

[25] David Hestenes and Garret Sobczyk. *Clifford algebra to geometric calculus: a unified language for mathematics and physics*, volume 5. Springer Science & Business Media, 2012.

[26] Herbert B Keller. *Numerical solution of two point boundary value problems*. SIAM, 1976.

[27] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[28] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2688–2697. PMLR, 10–15 Jul 2018.

[29] Leon Klein, Andreas Krämer, and Frank Noe. Equivariant flow matching. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[30] Jonas Köhler, Leon Klein, and Frank Noe. Equivariant flows: Exact likelihood generative learning for symmetric densities. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5361–5370. PMLR, 13–18 Jul 2020.

[31] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[33] Cong Liu, David Ruhe, Floor Eijkelboom, and Patrick Forré. Clifford group equivariant simplicial message passing networks. In *The Twelfth International Conference on Learning Representations*, 2024.

[34] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32, 2019.

[35] Yang Liu, Saeed Anwar, Liang Zheng, and Qi Tian. Gradnet image denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[36] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[38] Douglas Lundholm and Lars Svensson. Clifford algebra, geometric algebra, and applications. *arXiv preprint arXiv:0907.5356*, 2009.

[39] Pavlo Melnyk, Michael Felsberg, and Mårten Wadenbäck. Embed me if you can: A geometric perceptron. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1276–1284, October 2021.

[40] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2015.

[41] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

[42] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[43] Dean S Oliver and Yan Chen. Recent progress on reservoir history matching: a review. *Computational Geosciences*, 15(1):185–221, 2011.

[44] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017.

[45] 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, 2019.

[46] Simiao Ren, Willie Padilla, and Jordan Malof. Benchmarking deep inverse models over time, and the neural-adjoint method. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 38–48. Curran Associates, Inc., 2020.

[47] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[48] Reuven Y Rubinstein and Dirk P Kroese. The cross-entropy method: A unified approach to monte carlo simulation, randomized optimization and machine learning. *Information Science & Statistics, Springer Verlag, NY*, 2004.

[49] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[50] David Ruhe, Johannes Brandstetter, and Patrick Forré. Clifford group equivariant neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[51] David Ruhe, Jayesh K Gupta, Steven de Keninck, Max Welling, and Johannes Brandstetter. Geometric clifford algebra networks. *arXiv preprint arXiv:2302.06594*, 2023.

[52] Victor Garcia Satorras, Emiel Hoogeboom, Fabian Bernd Fuchs, Ingmar Posner, and Max Welling. E(n) equivariant normalizing flows. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[53] 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.

[54] Matthew Spellings. Geometric algebra attention networks for small point clouds, 2022.

[55] Olivier Talagrand and Philippe Courtier. Variational assimilation of meteorological observations with the adjoint vorticity equation. i: Theory. *Quarterly Journal of the Royal Meteorological Society*, 113(478):1311–1328, 1987.

[56] Jeroen Tromp, Carl Tape, and Qinya Liu. Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels. *Geophysical Journal International*, 160(1):195–216, 2005.

[57] Ian Vernon, Michael Goldstein, and Richard Bower. Galaxy formation: Bayesian history matching for the observable universe. *Statistical science*, pages 81–90, 2014.

[58] Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg New York, 1996.

[59] Daniel Williamson, Michael Goldstein, Lesley Allison, Adam Blaker, Peter Challenor, Laura Jackson, and Kuniko Yamazaki. History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble. *Climate dynamics*, 41(7):1703–1729, 2013.

[60] Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[61] Maksim Zhdanov, David Ruhe, Maurice Weiler, Ana Lucic, Johannes Brandstetter, and Patrick Forré. Clifford-steerable convolutional neural networks, 2024.

Appendix

A Differentiability of functions between Clifford spaces

A.1 Metric on Clifford space

Here, we detail the definition of metrics, which are equivalent to those for Euclidean spaces, defined on real vector spaces and show that the metrics on vector spaces induce metrics on their associated exterior algebras, following the definition of [18, 19, 20]. Let V be a finite-dimensional vector space in $\mathbb{R}$. Then, the exterior algebra $\bigwedge^* V$ of V is defined as the quotient of tensor algebra $\bigotimes V$ of V divided by an ideal generated by $v \otimes w - w \otimes v$, in which we denote $v \wedge w$ instead of $v \otimes w$. This leads to the anti-commutative relation $v \wedge w = w \wedge v$ in $\bigwedge^* V$. In order to introduce a metric on $\bigwedge^* V$, we first introduce Euclidean metric:

Definition A.1. *We call a mapping between* $\mathfrak{g}_V : V \times V \to \mathbb{R}$ *an* Euclidean metric *for* V *iff* $\mathfrak{g}_V$ *satisfies the following conditions:*

$$\begin{aligned}
&\mathfrak{g}_V(v, w) = \mathfrak{g}_V(w, v), \forall v, w \in V,\\
&\mathfrak{g}_V(v, w) = 0, \forall w \in V \Rightarrow v = 0,\\
&\mathfrak{g}_V(v, v) \geq 0, \forall v, w \in V, \quad if\ \mathfrak{g}_V(v, v) = 0,\ then\ v = 0.
\end{aligned}$$

The pair $(V, \mathfrak{g}_V)$ *is called an* Euclidean structure *for* V.

Now, associated to $(V, \mathfrak{g}_V)$, we define the scalar product of $\boldsymbol{x}, \boldsymbol{y} \in \bigwedge^* V$. For $\boldsymbol{x} \in \bigwedge^* V$, we denote the m-th fold exterior product composing $\boldsymbol{x}$ by

$$\boldsymbol{x}^{(m)} \in \bigwedge^m V = \underbrace{V \bigwedge \cdots \bigwedge V}_{m}.$$

Note that $\boldsymbol{x}^{(0)}$ belongs to a scalor field, i.e., $\boldsymbol{x}^{(0)} \in \mathbb{R}$.

Definition A.2. *A scalar product on* $\bigwedge^* V$ *is*

$$\mathfrak{g}_{\bigwedge^* V}(x, y) = x^{(0)} y^{(0)} + \sum_{k=1}^{n} \left(\frac{1}{k}\right)^2 \sum_{I_m, j_m} x_{I_m} y_{j_m} \det((\mathfrak{g}_V(e_i, e_j))_{i \in I_m, j \in J_m}),$$

in which $I_m = \{i_1, \cdots, i_m\}, J_m = \{j_1, \cdots, j_m\} \subset \{1, 2, \cdots, n\}$, *and* x_{I_m} *and* y_{I_m} *are coefficients for* e_{I_m} *and* e_{J_m}.

We note that this product is a well-defined and symmetric, non-degenerate, and positive definite, which means $\mathfrak{g}_{\bigwedge^* V}$ defines an Euclidean structure for $\bigwedge^* V$. The reason we have $\left(\frac{1}{k}\right)^2$ is that we would like the product to be invariant to the order of indices I_m. However, if we arrange the order of I_m, say ascending order as in decomposition of Clifford space Eq.(1), this expression has the following equivalent definition

$$\mathfrak{g}_{Cl(V,\mathfrak{q})}(x, y) = \sum_{m=0}^{n} \boldsymbol{x}^{(k)} \cdot \boldsymbol{y}^{(k)} = \sum_{m=0}^{n} \sum_{I_m} x_{I_m} y_{I_m} \det((\mathfrak{g}_V(\boldsymbol{e}_{i_k}, \boldsymbol{e}_{j_l}))_{i_k, j_l \in I_m}).$$

We note that since the metric $\mathfrak{g}_{Cl(V,\mathfrak{q})}$ is defined by the determinant of $(\mathfrak{g}_V(\boldsymbol{e}_{i_k}, \boldsymbol{e}_{j_l}))_{i_k, j_l \in I_m}$, which measures the magnitude of higher-dimensional grade elements as well as 1-dimensional vectors, this can be considered as an extension of the notion of differentiable functions between Euclidean spaces. For further details, we encourage the readers to have a look at [18, 19, 20].

A.2 Definition on differentiability

We first introduce subspaces of Clifford algebra. Eq.(1) defines vector subspaces $Cl^{(m)}(V, \mathfrak{q}) = \{\sum_{I_m} x_{I_m} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m} \mid I_m \subset \{1, 2, \cdots, n\},\ x_{I_m} \in \mathbb{R}\}$ for $m \in \{0, 1, \cdots, n\}$, called

m-*grades*. Therefore, the Clifford algebra $Cl(V,\mathfrak{q})$ has a canonical decomposition into the direct sum of $Cl^{(m)}(V,\mathfrak{q})$:

$$Cl(V,\mathfrak{q}) = \bigoplus_{m=0}^{n} Cl^{(m)}(V,\mathfrak{q}), \quad \dim_{\mathbb{R}} Cl^{(m)}(V,\mathfrak{q}) = \binom{n}{m}.$$

With this decomposition, we also write $\boldsymbol{x}^{(m)}$ for $\boldsymbol{x} \in Cl(V,\mathfrak{q})$ as the component of $\boldsymbol{x}$ belonging to $Cl^{(m)}(V,\mathfrak{q})$. We note that all the above settings and properties hold for any orthogonal basis of V, and Clifford algebra for V is unique up to isomorphism.

Let $\boldsymbol{F}$ be a function from $Cl^{(p)}(V,\mathfrak{q}) \to Cl^{(q)}(V,\mathfrak{q})$. $\boldsymbol{F}$ is said to be *differentiable at* $\boldsymbol{x}_0$ if there exists a function $\boldsymbol{f}_{\boldsymbol{x}_0} : Cl^{(p)}(V,\mathfrak{q}) \to Cl^{(q)}(V,\mathfrak{q})$ such that

$$\lim_{\boldsymbol{x}\to\boldsymbol{x}_0} \frac{\boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}_0) - \boldsymbol{f}_{\boldsymbol{x}_0}(\boldsymbol{x}-\boldsymbol{x}_0)}{||\boldsymbol{x}-\boldsymbol{x}_0||} = 0.$$

We note that using the triangular inequality, we can show that such $\boldsymbol{f}_{\boldsymbol{x}_0}$ is unique if it exists, see also [20]. Based on this definition, we can further derive the definition of directional derivative: Suppose $\boldsymbol{F}$ is differentiable at $\boldsymbol{x}_0$. It is known from [20] that for $\forall \boldsymbol{a} \in Cl^{(p)}(V,\mathfrak{q})$, we have

$$\lim_{\lambda\to 0} \frac{\boldsymbol{F}(\boldsymbol{x}_0 + \lambda\boldsymbol{a}) - \boldsymbol{F}(\boldsymbol{x}_0)}{\lambda} = \boldsymbol{f}_{\boldsymbol{x}_0}(\boldsymbol{a}).$$

Based on this fact, we define the $\boldsymbol{a}$-directional derivative of $\boldsymbol{F}$ (not necessarily differentiable) at $\boldsymbol{x}_0$, denoted by $\boldsymbol{F}'_{\boldsymbol{a}}(\boldsymbol{x}_0)$, to be

$$\boldsymbol{F}'_{\boldsymbol{a}}(\boldsymbol{x}_0) = \lim_{\lambda\to 0} \frac{\boldsymbol{F}(\boldsymbol{x}_0 + \lambda\boldsymbol{a}) - \boldsymbol{F}(\boldsymbol{x}_0)}{\lambda}.$$

We note that this definition is equivalent to that in Eq.(4). Namely, for $\forall \boldsymbol{F} : Cl(V,\mathfrak{q}) \to Cl(V,\mathfrak{q})$, we can always derive a function between $Cl^{(p)}(V,\mathfrak{q}) \to Cl^{(q)}(V,\mathfrak{q})$ by composing $\boldsymbol{F}$ with the canonical inclusion $Cl^{(p)}(V,\mathfrak{q}) \hookrightarrow Cl(V,\mathfrak{q})$ and projection map $Cl(V,\mathfrak{q}) \to Cl^{(q)}(V,\mathfrak{q})$. On the other hand, when we have two functions $\boldsymbol{F_1} : Cl^{(p_1)}(V,\mathfrak{q}) \to Cl^{(q_1)}(V,\mathfrak{q})$ and $\boldsymbol{F_2} : Cl^{(p_2)}(V,\mathfrak{q}) \to Cl^{(q_2)}(V,\mathfrak{q})$, by taking the direct product (i.e., concatenation), we have

$$(\boldsymbol{F_1},\boldsymbol{F_2}) : Cl^{(p_1)}(V,\mathfrak{q})\times Cl^{(p_2)}(V,\mathfrak{q}) \to Cl^{(q_1)}(V,\mathfrak{q})\times Cl^{(q_2)}(V,\mathfrak{q}), \quad (\boldsymbol{x}_1,\boldsymbol{x}_2) \mapsto (\boldsymbol{F_1}(\boldsymbol{x}_1),\boldsymbol{F_2}(\boldsymbol{x}_2)).$$

The notion of (directional) differentiability is also equivalent to Definition 4, since the product of canonical Euclidean distances d_{n_1} and d_{n_2} of two Euclidean spaces $\mathbb{R}^{n_1}$ and $\mathbb{R}^{n_2}$ is equivalent to the canonical distance $d_{n_1+n_2}$ of $\mathbb{R}^{n_1+n_2}$.

B Analysis of functions between Clifford algebras

Coordinate system For the continuous normalizing flow experiments, we need the definition of probability distribution on the Clifford Algebra. Since the metric, defined on the Clifford Algebra through their bilinear form, can be degenerate and not positive, we resort to the Euclidean metric defined on the Euclidean coordinate system. We first define two mappings as follows:

$$\mathrm{inc}_{\mathsf{coord}} : \mathbb{R}^{2^n} \hookrightarrow Cl(\mathbb{R}^n,\mathfrak{q}), \boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \mapsto \sum_{m=0}^{n} \sum_{I_m=\{i_1\ldots i_m\}} x_{I_m}\boldsymbol{e}_{I_m}, \tag{6}$$

$$\mathrm{proj}_{\mathsf{coord}} : Cl(\mathbb{R}^n,\mathfrak{q}) \twoheadrightarrow \mathbb{R}^{2^n}, \boldsymbol{x} = \sum_{m=0}^{n} \sum_{I_m} x_{I_m}\boldsymbol{e}_{I_m} \mapsto \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix}, \tag{7}$$

where $\boldsymbol{e}_{I_m} = \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m}$ and $I_m = \{i_1 < \cdots < i_m\} \subset \{1,2,\cdots,n\}$. Note that $I_m = \emptyset$ for $m = 0$. We here suppose that the metric on the Clifford algebra is defined as the bilinear form $\mathfrak{b}$ with

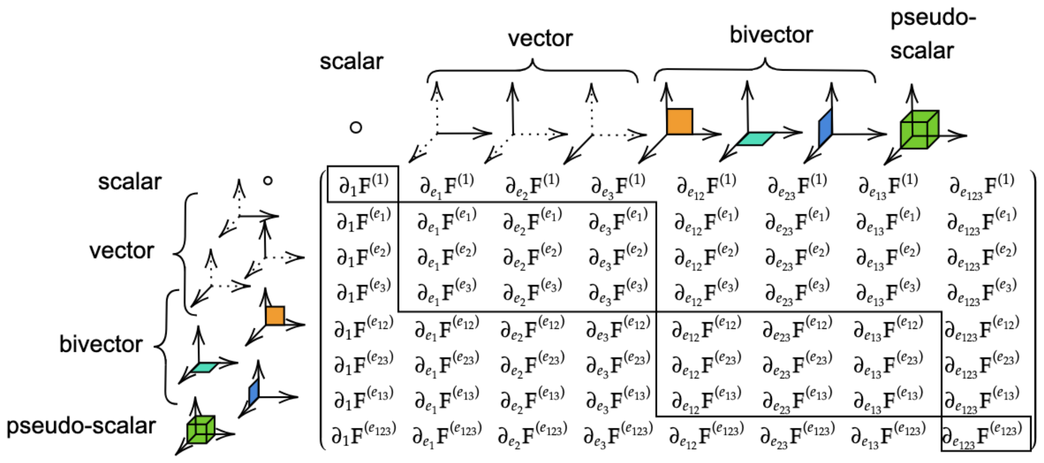
scalar
vector
bivector
pseudo-scalar
scalar
vector
bivector
pseudo-scalar
$\partial_1 F^{(1)}$ $\partial_{e_1} F^{(1)}$ $\partial_{e_2} F^{(1)}$ $\partial_{e_3} F^{(1)}$ $\partial_{e_{12}} F^{(1)}$ $\partial_{e_{23}} F^{(1)}$ $\partial_{e_{13}} F^{(1)}$ $\partial_{e_{123}} F^{(1)}$
$\partial_1 F^{(e_1)}$ $\partial_{e_1} F^{(e_1)}$ $\partial_{e_2} F^{(e_1)}$ $\partial_{e_3} F^{(e_1)}$ $\partial_{e_{12}} F^{(e_1)}$ $\partial_{e_{23}} F^{(e_1)}$ $\partial_{e_{13}} F^{(e_1)}$ $\partial_{e_{123}} F^{(e_1)}$
$\partial_1 F^{(e_2)}$ $\partial_{e_1} F^{(e_2)}$ $\partial_{e_2} F^{(e_2)}$ $\partial_{e_3} F^{(e_2)}$ $\partial_{e_{12}} F^{(e_2)}$ $\partial_{e_{23}} F^{(e_2)}$ $\partial_{e_{13}} F^{(e_2)}$ $\partial_{e_{123}} F^{(e_2)}$
$\partial_1 F^{(e_3)}$ $\partial_{e_1} F^{(e_3)}$ $\partial_{e_2} F^{(e_3)}$ $\partial_{e_3} F^{(e_3)}$ $\partial_{e_{12}} F^{(e_3)}$ $\partial_{e_{23}} F^{(e_3)}$ $\partial_{e_{13}} F^{(e_3)}$ $\partial_{e_{123}} F^{(e_3)}$
$\partial_1 F^{(e_{12})}$ $\partial_{e_1} F^{(e_{12})}$ $\partial_{e_2} F^{(e_{12})}$ $\partial_{e_3} F^{(e_{12})}$ $\partial_{e_{12}} F^{(e_{12})}$ $\partial_{e_{23}} F^{(e_{12})}$ $\partial_{e_{13}} F^{(e_{12})}$ $\partial_{e_{123}} F^{(e_{12})}$
$\partial_1 F^{(e_{23})}$ $\partial_{e_1} F^{(e_{23})}$ $\partial_{e_2} F^{(e_{23})}$ $\partial_{e_3} F^{(e_{23})}$ $\partial_{e_{12}} F^{(e_{23})}$ $\partial_{e_{23}} F^{(e_{23})}$ $\partial_{e_{13}} F^{(e_{23})}$ $\partial_{e_{123}} F^{(e_{23})}$
$\partial_1 F^{(e_{13})}$ $\partial_{e_1} F^{(e_{13})}$ $\partial_{e_2} F^{(e_{13})}$ $\partial_{e_3} F^{(e_{13})}$ $\partial_{e_{12}} F^{(e_{13})}$ $\partial_{e_{23}} F^{(e_{13})}$ $\partial_{e_{13}} F^{(e_{13})}$ $\partial_{e_{123}} F^{(e_{13})}$
$\partial_1 F^{(e_{123})}$ $\partial_{e_1} F^{(e_{123})}$ $\partial_{e_2} F^{(e_{123})}$ $\partial_{e_3} F^{(e_{123})}$ $\partial_{e_{12}} F^{(e_{123})}$ $\partial_{e_{23}} F^{(e_{123})}$ $\partial_{e_{13}} F^{(e_{123})}$ $\partial_{e_{123}} F^{(e_{123})}$

Figure 2: *Clifford Jacobian* ($\boldsymbol{J} = \{\partial_I \boldsymbol{F}^{(J)}\}_{I,J}$) with respect to the coordinate system $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$ and its derived blades for a Clifford algebra with $n = 3$. The Clifford directional gradient is defined as $\partial_I \boldsymbol{F}^{(J)} = \lim_{\lambda\to 0} \frac{\boldsymbol{F}^{(J)}(\boldsymbol{x}+\lambda \boldsymbol{e}_I)-\boldsymbol{F}^{(J)}(\boldsymbol{x})}{\lambda}$. Highlighted in the boxes the gradient of the scalar components with respect to the scalar component, the vector, bivector and pseudo-scalar Jacobians. The element of the Clifford Jacobian are $\partial_I \boldsymbol{F}^{(J)}$, where each column highlight the direction $\boldsymbol{e}_I$ of the gradient, while the row indicates the component $\boldsymbol{e}_J$ at the output of the function.

signature $(n, 0, 0)$. Then, the above two functions turn out to be isometric functions, and we get the following commutative diagram (including dotted backward arrows)

$$\begin{tikzcd}
Cl(\mathbb{R}^n, \mathfrak{q}) \arrow[r, "\boldsymbol{F}"] \arrow[d, dotted, bend right] & Cl(\mathbb{R}^n, \mathfrak{q}) \arrow[d, "\mathrm{proj}_{\text{coord}}"] \arrow[l, dotted, bend right] \\
\mathbb{R}^{2^n} \arrow[u, "\mathrm{inc}_{\text{coord}}"] \arrow[r, "\boldsymbol{f}"'] & \mathbb{R}^{2^n} \arrow[l, dotted, bend right] \arrow[u, dotted, bend right]
\end{tikzcd}$$

Here $\boldsymbol{f}$ is defined as $\boldsymbol{f} = \mathrm{proj}_{\text{coord}} \circ \boldsymbol{F} \circ \mathrm{inc}_{\text{coord}} : \mathbb{R}^{2^n} \mapsto \mathbb{R}^{2^n} : \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x})$. This function is $\boldsymbol{a}$-directional differentiable in the usual sense, when $\boldsymbol{F}$ is differentiable.

Jacobian matrix of differentiable Clifford functions We define the Jacobian of functions between Clifford algebras via the directional gradient. Given a differentiable function $\boldsymbol{F} : Cl(\mathbb{R}^n, \mathfrak{q}) \to Cl(\mathbb{R}^n, \mathfrak{q})$, the directional gradient of the J-th component $\boldsymbol{F}^{(J)}$ in the output space along the direction $\boldsymbol{e}_I$ in the input space is defined as follows:

$$\partial_I \boldsymbol{F}^{(J)} = \lim_{\lambda\to 0} \frac{\boldsymbol{F}^{(J)}(\boldsymbol{x} + \lambda \boldsymbol{e}_I) - \boldsymbol{F}^{(J)}(\boldsymbol{x})}{\lambda} \in Cl(\mathbb{R}^n, \mathfrak{q}), \tag{8}$$

$$\partial_I \boldsymbol{F} = \sum_J \partial_I \boldsymbol{F}^{(J)} \in Cl(\mathbb{R}^n, \mathfrak{q}). \tag{9}$$

We define the Jacobian of $\boldsymbol{F}$ as

$$\boldsymbol{J}_{\boldsymbol{F}} = (\partial_I \boldsymbol{F})_I = \begin{pmatrix} \partial_1 \boldsymbol{F} \\ \vdots \\ \partial_{2^n} \boldsymbol{F} \end{pmatrix} = \begin{pmatrix} \partial_1 \boldsymbol{F}^{(1)} + \cdots + \partial_1 \boldsymbol{F}^{(2^n)} \\ \vdots \\ \partial_{2^n} \boldsymbol{F}^{(1)} + \cdots + \partial_{2^n} \boldsymbol{F}^{(2^n)} \end{pmatrix} \in Cl(\mathbb{R}^n, \mathfrak{q})^{2^n}, \tag{10}$$

which is equivalent to the Jacobian of f in the coordinate system through $\mathrm{proj}_{\text{coord}}$ and $\mathrm{inc}_{\text{coord}}$:

$$\boldsymbol{J}_{\boldsymbol{f}} = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} = \begin{pmatrix} \partial_1 \boldsymbol{f}^{(1)}, \ldots, \partial_1 \boldsymbol{f}^{(2^n)} \\ \vdots \\ \partial_{2^n} \boldsymbol{f}^{(1)}, \ldots, \partial_{2^n} \boldsymbol{f}^{(2^n)} \end{pmatrix} \in \mathbb{R}^{2^n \times 2^n}. \tag{11}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{dy}}^T \frac{\mathrm{df(x)}}{\mathrm{dx}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{dy}}^T \mathrm{J_f(x)} = \overbrace{\begin{pmatrix} \partial_1\mathcal{L} \\ \partial_{e_1}\mathcal{L} \\ \partial_{e_2}\mathcal{L} \\ \partial_{e_3}\mathcal{L} \\ \partial_{e_{12}}\mathcal{L} \\ \partial_{e_{23}}\mathcal{L} \\ \partial_{e_{13}}\mathcal{L} \\ \partial_{e_{123}}\mathcal{L} \end{pmatrix}^T}^{\frac{\mathrm{d}\mathcal{L}}{\mathrm{dy}}^T} \cdot \overbrace{\begin{pmatrix}
\partial_1\mathbf{f}^{(1)} & \partial_{e_1}\mathbf{f}^{(1)} & \partial_{e_2}\mathbf{f}^{(1)} & \partial_{e_3}\mathbf{f}^{(1)} & \partial_{e_{12}}\mathbf{f}^{(1)} & \partial_{e_{23}}\mathbf{f}^{(1)} & \partial_{e_{13}}\mathbf{f}^{(1)} & \partial_{e_{123}}\mathbf{f}^{(1)} \\
\partial_1\mathbf{f}^{(e_1)} & \partial_{e_1}\mathbf{f}^{(e_1)} & \partial_{e_2}\mathbf{f}^{(e_1)} & \partial_{e_3}\mathbf{f}^{(e_1)} & \partial_{e_{12}}\mathbf{f}^{(e_1)} & \partial_{e_{23}}\mathbf{f}^{(e_1)} & \partial_{e_{13}}\mathbf{f}^{(e_1)} & \partial_{e_{123}}\mathbf{f}^{(e_1)} \\
\partial_1\mathbf{f}^{(e_2)} & \partial_{e_1}\mathbf{f}^{(e_2)} & \partial_{e_2}\mathbf{f}^{(e_2)} & \partial_{e_3}\mathbf{f}^{(e_2)} & \partial_{e_{12}}\mathbf{f}^{(e_2)} & \partial_{e_{23}}\mathbf{f}^{(e_2)} & \partial_{e_{13}}\mathbf{f}^{(e_2)} & \partial_{e_{123}}\mathbf{f}^{(e_2)} \\
\partial_1\mathbf{f}^{(e_3)} & \partial_{e_1}\mathbf{f}^{(e_3)} & \partial_{e_2}\mathbf{f}^{(e_3)} & \partial_{e_3}\mathbf{f}^{(e_3)} & \partial_{e_{12}}\mathbf{f}^{(e_3)} & \partial_{e_{23}}\mathbf{f}^{(e_3)} & \partial_{e_{13}}\mathbf{f}^{(e_3)} & \partial_{e_{123}}\mathbf{f}^{(e_3)} \\
\partial_1\mathbf{f}^{(e_{12})} & \partial_{e_1}\mathbf{f}^{(e_{12})} & \partial_{e_2}\mathbf{f}^{(e_{12})} & \partial_{e_3}\mathbf{f}^{(e_{12})} & \partial_{e_{12}}\mathbf{f}^{(e_{12})} & \partial_{e_{23}}\mathbf{f}^{(e_{12})} & \partial_{e_{13}}\mathbf{f}^{(e_{12})} & \partial_{e_{123}}\mathbf{f}^{(e_{12})} \\
\partial_1\mathbf{f}^{(e_{23})} & \partial_{e_1}\mathbf{f}^{(e_{23})} & \partial_{e_2}\mathbf{f}^{(e_{23})} & \partial_{e_3}\mathbf{f}^{(e_{23})} & \partial_{e_{12}}\mathbf{f}^{(e_{23})} & \partial_{e_{23}}\mathbf{f}^{(e_{23})} & \partial_{e_{13}}\mathbf{f}^{(e_{23})} & \partial_{e_{123}}\mathbf{f}^{(e_{23})} \\
\partial_1\mathbf{f}^{(e_{13})} & \partial_{e_1}\mathbf{f}^{(e_{13})} & \partial_{e_2}\mathbf{f}^{(e_{13})} & \partial_{e_3}\mathbf{f}^{(e_{13})} & \partial_{e_{12}}\mathbf{f}^{(e_{13})} & \partial_{e_{23}}\mathbf{f}^{(e_{13})} & \partial_{e_{13}}\mathbf{f}^{(e_{13})} & \partial_{e_{123}}\mathbf{f}^{(e_{13})} \\
\partial_1\mathbf{f}^{(e_{123})} & \partial_{e_1}\mathbf{f}^{(e_{123})} & \partial_{e_2}\mathbf{f}^{(e_{123})} & \partial_{e_3}\mathbf{f}^{(e_{123})} & \partial_{e_{12}}\mathbf{f}^{(e_{123})} & \partial_{e_{23}}\mathbf{f}^{(e_{123})} & \partial_{e_{13}}\mathbf{f}^{(e_{123})} & \partial_{e_{123}}\mathbf{f}^{(e_{123})}
\end{pmatrix}}^{\mathrm{J_f(x)}}$$

Figure 3: Example of *vector-Jacobian product* of functions between Clifford algebras with $n = 3$, when computing the loss for a machine learning task.

Then, this gradient is compatible with gradient on functions between Clifford algebras through the Jacobian of $\mathrm{proj}_{\mathrm{coord}}$ and $\mathrm{inc}_{\mathrm{coord}}$, that is:

$$\boldsymbol{J_f} = \boldsymbol{J}_{\mathrm{inc}_{\mathrm{coord}}} \boldsymbol{J_F} \boldsymbol{J}_{\mathrm{proj}_{\mathrm{coord}}}.$$

Vector-Jacobian product When we want to implement the chain rule, we proceed analogously as in the Euclidean case. For example let suppose as in machine learning tasks, we have a loss function $\mathcal{L}(y)$ of the output variable of a Clifford neural network $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, then we are interested in the variation of the loss for the variation of the input variable, i.e.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{x}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{y}}^T \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{y}}^T \boldsymbol{J}_{\mathbf{f}}(\boldsymbol{x}) \tag{12}$$

as shown in Figure 3.

Density functions over Clifford algebra Since the Clifford algebra $Cl(\mathbb{R}^n, \mathfrak{q})$ is equipped with the Euclidean scalar metric $\mathfrak{g}_{Cl(\mathbb{R}^n,\mathfrak{q})}$, we have a measure $\mu(\boldsymbol{x})$ on $Cl(\mathbb{R}^n, \mathfrak{q})$, that is equivalent to the canonical measure on $\mathbb{R}^{2^n}$. Through this measure, we define a probability density function $p(\boldsymbol{x})$ on $Cl(\mathbb{R}^n, \mathfrak{q})$ such that

$$\int_{Cl(\mathbb{R}^n,\mathfrak{q})} p(\boldsymbol{x}) d\mu(\boldsymbol{x}) = 1.$$

We can then also build the same probability theory on the space of $Cl(\mathbb{R}^n, \mathfrak{q})$ as the Euclidean space. Section E employs this probability theory when performing CNF in the space of Clifford space. Namely, on the space of $Cl(\mathbb{R}^n, \mathfrak{q})$, we have two equivalent formulas

$$\ln p(\boldsymbol{X}_1) = \ln p(\boldsymbol{X}_0) - \ln \left| \det \frac{\partial \boldsymbol{F}(\boldsymbol{x}_0)}{\partial \boldsymbol{X}} \right|, \tag{13}$$

where $\boldsymbol{X}_1 = \boldsymbol{F}(\boldsymbol{X}_0)$ based on the change of variable formula on $Cl(\mathbb{R}^n, \mathfrak{q})$.

C Differentiability of Clifford neural networks

We give a regorous statement of the differentiability of Clifford neural networks. The result can be viewed as an extension of the results in [14, 24], since the literature assume the case of $r = 0$.

Proposition C.1. *Let $p, q, r \in \mathbb{N}$ s.t. $p + q + r = n$. Then,*

(i) any $\boldsymbol{F} \in \mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ is differentiable.

(ii) For $\forall \boldsymbol{F}, \boldsymbol{G} \in \mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ and $\forall \boldsymbol{a} \in Cl(V, \mathfrak{q})$, the Leibniz rule holds:

$$(\boldsymbol{F} \otimes_{\mathfrak{q}} \boldsymbol{G})'_{\boldsymbol{a}}(X) = (\boldsymbol{F}'_{\boldsymbol{a}} \otimes_{\mathfrak{q}} \boldsymbol{G})(X) + (\boldsymbol{F} \otimes_{\mathfrak{q}} \boldsymbol{G}'_{\boldsymbol{a}})(X), \quad X = (X_1, X_2, \cdots, X_c).$$

Our definition of gradient adopts the bilinear form $\mathfrak{b}$ with signature $(n, 0, 0)$ as its metric $\mathfrak{g}_V$, even when taking the gradient of polynomials $\boldsymbol{F} \in \mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ with arbitrary p, q, and r. This is because the gradient defined by a metric associated with $\mathfrak{b}$ with either $q > 0$ or $r > 0$ is ill-defined (since $\mathfrak{b}$ is not positive-definite when $q > 0$ or $r > 0$.) We exposit problematic behaviors of gradients in a case of non positive-defininte metric in Appendix D. Our proposition implies that the gradient of $\boldsymbol{F} \in \mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ can also derive Jacobian of $\boldsymbol{F}$ equivalent to that of a function between Euclidean space of dimension 2^n, which we also detail in Appendix B.

While we are aware of the fact that the proof for majority of the claims in Proposition C.1 is essentially same as the proof for differentiable functions of the Euclidean space of dimension 2^n, we still give a comprehensive proof for all the claims.

Proof. (i) All the arithmetic operations involved to define polynomials $\mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ are the summation and geometric product. Therefore, we reduce the proof for (i) to the proof for the summation and geometric product. We here show the differentiability of $\boldsymbol{F}$ for each of the cases.

Summation. With Eq. (1), the sum of two elements $\boldsymbol{x}, \boldsymbol{y} \in Cl(V, \mathfrak{q})$ may be written as

$$\boldsymbol{x} + \boldsymbol{y} = \sum_{m=0}^{n} \sum_{I_m} (x_{I_m} + y_{I_m}) \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m}, \quad x_{I_m},\ y_{I_m} \in \mathbb{R}.$$

Let $\boldsymbol{a}$ be a constant element of $Cl(V, \mathfrak{q})$. The $\boldsymbol{a}$-directional gradient of $\boldsymbol{F}$ (with respect to $\boldsymbol{x}$) is

$$(\boldsymbol{x} + \boldsymbol{y})'_{\boldsymbol{a}} = \lim_{\lambda \to 0} \frac{(\boldsymbol{x} + \lambda \boldsymbol{a} + \boldsymbol{y}) - (\boldsymbol{x} + \boldsymbol{y})}{\lambda} = \lim_{\lambda \to 0} \frac{\lambda \boldsymbol{a}}{\lambda} = \boldsymbol{a}.$$

Geometric product. This proof is further reduced to the case of the geometric product of each pair of basis elements $\boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_m}$ composing respective $\boldsymbol{x}, \boldsymbol{y} \in Cl(V, \mathfrak{q})$, since each element in $Cl(\mathbb{R}^n, \mathfrak{q})$ has the decomposition in Eq. (1) and the product runs over all pairs of the basis. Namely,

$$\begin{aligned}
\boldsymbol{x} \otimes_{\mathfrak{q}} \boldsymbol{y} &= \left(\sum_{r=0}^{n} \sum_{I_r} x_{I_r} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r} \right) \otimes_{\mathfrak{q}} \left(\sum_{s=0}^{n} \sum_{J_s} y_{J_s} \boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s} \right) \\
&= \sum_{r,s=0}^{n} \sum_{I_r, J_s} \left(x_{I_r} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r} \right) \otimes_{\mathfrak{q}} \left(y_{J_s} \boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s} \right). \qquad (14)
\end{aligned}$$

Noting Eq. (2), the geometric product of each pair of basis elements may be written as

$$\left(x_{I_r} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r} \right) \otimes_{\mathfrak{q}} \left(y_{J_s} \boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s} \right) = x_{I_r} y_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u}) (\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}}). \qquad (15)$$

Therefore, for $\boldsymbol{a} \in Cl(V, \mathfrak{q})$, the $\boldsymbol{a}$-directional derivative of the geometric product with respect to $\boldsymbol{x}$ is

$$\begin{aligned}
&(\boldsymbol{x} \otimes_{\mathfrak{q}} \boldsymbol{y})'_{\boldsymbol{a}} \\
&= \lim_{\lambda \to 0} \frac{\left(\sum_{r=0}^{n} \sum_{I_r} (x_{I_r} + \lambda a_{I_r}) \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r} \right) \otimes_{\mathfrak{q}} \left(\sum_{s=0}^{n} \sum_{J_s} y_{J_s} \boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s} \right)}{\lambda} \\
&\quad - \frac{\left(\sum_{r=0}^{n} \sum_{I_r} x_{I_r} \boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r} \right) \otimes_{\mathfrak{q}} \left(\sum_{s=0}^{n} \sum_{J_s} y_{J_s} \boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s} \right)}{\lambda} \\
&= \lim_{\lambda \to 0} \frac{\sum_{r,s=0}^{n} \sum_{I_r, J_s} \lambda a_{I_r} y_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u}) (\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&= \lim_{\lambda \to 0} \sum_{r,s=0}^{n} \sum_{I_r, J_s} a_{I_r} y_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u}) (\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}}) \\
&= \sum_{r,s=0}^{n} \sum_{I_r, J_s} a_{I_r} y_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u}) (\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}}).
\end{aligned}$$

(ii) Recall that any polynomial in $\mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$ is the linear sum of monomials in $\mathbb{R}[X_1, X_2, \cdots, X_c]_{p,q,r}$. Noting that the geometric product is bilinear and associative, it suffices to show the proof for the case of the geometric product of each pair of the basis elements as shown in Eq. (1). When we denote $\boldsymbol{F}(\boldsymbol{x}) = x_{I_r}\boldsymbol{e}_{i_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{i_r}$, and $\boldsymbol{G}(\boldsymbol{x}) = x_{J_s}\boldsymbol{e}_{j_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{j_s}$, the geometric product of $\boldsymbol{F}$ and $\boldsymbol{G}$ is

$$(\boldsymbol{F} \otimes_{\mathfrak{q}} \boldsymbol{G})(\boldsymbol{x}) = x_{I_r}x_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}}).$$

Therefore, for $\boldsymbol{a} \in Cl(V, \mathfrak{q})$, we get

$$\begin{aligned}
&(\boldsymbol{F}\otimes_{\mathfrak{q}}\boldsymbol{G})'_{\boldsymbol{a}}(\boldsymbol{x}) \\
&= \lim_{\lambda\to 0} \frac{(x_{I_r} + \lambda a_{I_r})(x_{J_s} + \lambda a_{J_s}) \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&\qquad - \frac{x_{I_r}x_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&= \lim_{\lambda\to 0} \frac{\big(\lambda(a_{I_r}x_{J_s} + a_{J_s}x_{I_r}) + \lambda^2 a_{I_r}a_{J_s}\big) \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&= \lim_{\lambda\to 0} \frac{\lambda a_{I_r}x_{J_s} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&\qquad + \lim_{\lambda\to 0} \frac{\lambda a_{J_s}x_{I_r} \prod_{u=0}^{t-1} \mathfrak{q}(\boldsymbol{e}_{r+s-u})(\boldsymbol{e}_{k_1} \otimes_{\mathfrak{q}} \cdots \otimes_{\mathfrak{q}} \boldsymbol{e}_{k_{r+s-t}})}{\lambda} \\
&= (\boldsymbol{F}'_{\boldsymbol{a}} \otimes_{\mathfrak{q}} \boldsymbol{G})(\boldsymbol{x}) + (\boldsymbol{F} \otimes_{\mathfrak{q}} \boldsymbol{G}'_{\boldsymbol{a}})(\boldsymbol{x}).
\end{aligned}$$

□

D Theoretical analysis on the gradient of Clifford neural networks

D.1 What if the metric is not positive?

We here give an example of ill-defined gradients which can happen in case we have non-positive metric. Let us again first detail the derivation of the gradient of a function $F: x_0 + x_1e_1 + x_2e_2 + x_{12}e_{12} \to x_1e_1 + x_2e_2$ along a direction $a = e_1 + e_2$: Suppose the gradient of F is $L = l_0 + l_1e_1 + l_2e_2 + l_{12}e_{12}$. Recall that the gradient of a function between Clifford algebras is defined as

$$\lim_{\lambda\to 0} \frac{F(x_0 + \lambda a) - F(x_0)}{\lambda}.$$

Then, we have the following equivalent equations:

$$\begin{aligned}
&\lim_{\lambda\to 0} \frac{F(x_0 + \lambda a) - F(x_0)}{\lambda} = L \\
\Leftrightarrow &\lim_{\lambda\to 0} \frac{(F(x_0 + \lambda a) - F(x_0)) - \lambda L}{\lambda} = 0 \\
\Leftrightarrow &\lim_{\lambda\to 0} \frac{(x_1 + \lambda)e_1 + (x_2 + \lambda)e_2 - (x_1e_1 + x_2e_2) - \lambda(l_0 + l_1e_1 + l_2e_2 + l_{12}e_{12})}{\lambda} = 0 \\
\Leftrightarrow &\lim_{\lambda\to 0} \frac{\lambda e_1 + \lambda e_2 - \lambda(l_0 + l_1e_1 + l_2e_2 + l_{12}e_{12})}{\lambda} = 0 \\
\Leftrightarrow &\lim_{\lambda\to 0} -l_0 + (1 - l_1)e_1 + (1 - l_2)e_2 - l_{12}e_{12} = 0 \cdots (a)
\end{aligned}$$

Then, by the definition of the limit operation $\lim_{\lambda\to 0}$, Eq. (a) is rewritten as the ϵ-δ definition of limit as follows: For $\forall\epsilon > 0$, there exists $\delta > 0$ such that

$$\lambda < \delta \Rightarrow \|-l_0 + (1 - l_1)e_1 + (1 - l_2)e_2 - l_{12}e_{12}\| < \epsilon. \cdots (b)$$

Since Eq. (a) does not include λ, (b) is equivalent to

$$\|-l_0 + (1 - l_1)e_1 + (1 - l_2)e_2 - l_{12}e_{12}\| = 0. \cdots (c)$$

Let us now assume that the norm $\|\cdot\|$ on the Clifford algebra is induced by the quadratic form $\mathfrak{q}$, i.e., $\|x\|^2 = g_{Cl(\mathbb{R}^2,\mathfrak{q})}(x,x)$, as in Section 3.1 in the main text and Appendix A.2. Then, the (squared) norm for a multivector $x = x_0 + x_1e_1 + x_2e_2 + x_{12}e_{12}$ may be written as

$$\|x_0 + x_1e_1 + x_2e_2 + x_{12}e_{12}\|^2 = x_0^2 + \mathfrak{q}(e_1)x_1^2 + \mathfrak{q}(e_2)x_2^2 + \mathfrak{q}(e_1)\mathfrak{q}(e_2)x_{12}^2.$$

Therefore, when $\mathfrak{q}(e_1) = \mathfrak{q}(e_2) = 1$, (c) is reduced to

$$l_0^2 + (l_1 - 1)^2 + (l_2 - 1)^2 + l_{12}^2 = 0.$$

Then, we get a unique solution $l_0 = 0, l_1 = 1, l_2 = 1, l_{12} = 0$, and therefore the gradient L is uniquely determined in this case. On the other hand, when $\mathfrak{q}(e_1) = -\mathfrak{q}(e_2) = 1$, we get

$$l_0^2 + (l_1 - 1)^2 - (l_2 - 1)^2 - l_{12}^2 = 0.$$

This equation has multiple solutions L, which means the gradient L is not well-defined.

D.2 Disadvantage using ill-defined gradients in applications.

In case we use gradients defined on a non-Euclidean metric, it is inevitable that we face difficulty in the treatment of the gradients. We give such an example in this subsection. We consider the root finding problem with the classical Newton method. Our objective is: For a given function $f(x)$, find a solution x of $f(x) = 0$. The Newton method in this case is written as

$$f(x_0 + a) = f(x_0) + a \cdot \partial_x f(x)|_{x_0} \cdots (d)$$

where $a \cdot \partial_x f(x)|_{x_0}$ is the directional gradient evaluated at x_0 along the direction a. Here, for simplicity, we assume $x_0, a \in \mathbb{R}^2$ and a problem in which the roots can be found with one-shot. What we want to find is the direction a that satisfies $f(x_0 + a) = 0$, therefore the equation above may be rewritten as

$$0 = f(x_0) + a \cdot \partial_x f(x)|_{x_0}, \cdots (e)$$

to set $f(x_0 + a) = 0$, $x_1 = x_0 + a$. We now take the identity function $f(x) = x$ and assume that the metric $||\cdot||$ on $\mathbb{R}^2$ is $||(x,y)|| = x^2 - y^2$, which essentially corresponds to $(p,q,r) = (1,1,0)$. For the directional derivative such that $a \cdot \partial_x f(x)|_{x_0} = L$, we have the ϵ-δ expression of the definition

$$\lim_{\epsilon \to 0} \epsilon^{-1}(f(x_0 + \epsilon a) - f(x_0) - \epsilon L) = 0.$$

A straightforward calculation leads to

$$\|a - L\|^2 = 0,$$

and by definition of the norm this equation is reduced to

$$(a_1 - l_1)^2 - (a_2 - l_2)^2 = 0$$
$$\Leftrightarrow l_1 = a_1 \pm |a_2 - l_2|.$$

From the expression, we can consider l_1 depends on l_2. By applying the result to (e), we have

$$0 = x_0 + (a_1 \pm |a_2 - l_2|, l_2) = (x_{0,1} + a_1 \pm |a_2 - l_2|, x_{0,2} + l_2).$$

Then, $l_2 = -x_{0,2}$. From the first entry in the above equation, we have an equation

$$0 = x_{0,1} + a_1 \pm |a_2 + x_2| \cdots (f)$$

that gives solutions (a_1, a_2). This gives you the following "solution" for $f(x_0 + a) = 0$:

$$x_0 + a = (\pm|a_2 + x_{0,2}|, x_{0,2} + a_2)$$

While $x_0 + a$ can be $(0,0)$, the above "solution" can have infinitely many choices which do not give $f(x_0 + a) = 0$.

This example showcases a possibility that the non-unique gradients can lead to wrong "solutions". This problem can be also observed in gradient descent algorithms: We suppose we want to find the minimum of $\min_x \mathcal{L}(f(x))$, with $\mathcal{L}(y)$ the loss function and $y = f(x)$. The gradient descent step is written as

$$x_1 = x_0 + a \cdot \partial_x f(x)|_{x_0} \cdots (g)$$
$$a = \partial_y \mathcal{L}(y)|_{y_0}$$

Here, we set $x_1 = x_0 + g$ with $g = a \cdot \partial_x f(x)|_{x_0}$ and $y_0 = f(x_0)$. We again use $f(x) = x$. Then, the directional derivative is $a \cdot \partial_x f(x)|_{x_0} = L$, and satisfies

$$\lim_{\epsilon \to 0} \epsilon^{-1}(f(x_0 + \epsilon a) - f(x_0) - \epsilon L) = 0$$

If we assume the norm is induced from $(p, q, r) = (1, 1, 0)$, by the $\epsilon - \delta$ definition, we get

$$\|a - L\|^2 = 0$$

that is reduced to

$$l_1 = a_1 \pm |a_2 - l_2|$$

We set l_2 as an independent variable, and then l_1 as a dependent variable. We now compute Eq. (g) with the above result:

$$x_1 = x_0 + (a_1 \pm |a_2 - l_2|, l_2)$$

per component:

$$x_{1,1} = x_{0,1} + a_1 \pm |a_2 - l_2|$$
$$x_{1,2} = x_{0,2} + l_2$$

Even if we consider a to be unique, the update x_1 is not unique and depends on l_2.

These examples demonstrate the difficulty of taking gradients, even for an elementary function, when the metric is defined with a general signature since the gradients cannot be uniquely determined, and it is not trivial to choose appropriate gradients. This fact, therefore, necessitates us to assume a non-degenerate metric in the Clifford algebra. Using autograd, we can avoid the above problems and get the proper gradient. Our experiments in the main text are meant to (indirectly) evaluate the properness of the gradient computed by Autograd by showing that the performance in respective tasks is reasonable.

E Sampling from probability distributions

Experimental setup. To evaluate the use of gradients in the Clifford Algebra for CNFs, we consider a Double Well (DW) and Lennard-Jones (LJ) particle systems, as presented in [30], which model the interactions among particles. These datasets have been generated by sampling from the system energy (see also Appendix G.2) using Markov Chain Monte Carlo sampling. **DW4** consists of four particles moving in a 2 dimensional space whose energy depends on a pair of particles. These systems have multiple metastable states. **LJ13** consists of 13 particles and models the potential between molecules as *Lennard-Jones* potentials, which describe long-range interactions and include both repulsive and attractive components. Following the experimental setup of [53], we use 10^3 samples for testing and validation, while the training is performed on $10, 10^2$ and 10^3 samples. We compare state of art E(n)-equivariant flow architectures, as "Simple dynamics", "Kernel dynamics" as described in [30]. Further, we consider Graph Normalizing Flow (GNF, [34]), with attention (GNF-att), and with attention and data augmentation (GNF-att-aug), i.e. when we augment the training data with rotations, and Equivariant Normalizing Flow (E-NF) as proposed in [53]. GNFs and E-NF consists of 3 layers and 32 features per layer, with SiLU activation function.

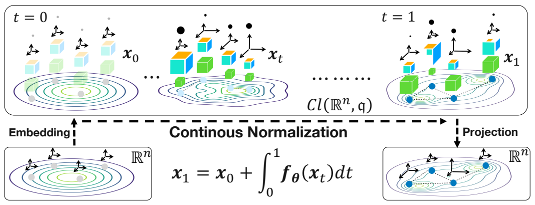

Figure 4: **Schematic of Continuous Normalizing Flow method**. The samples are generated starting from a random noise $\boldsymbol{x}_0 \sim N(0, \boldsymbol{I})$ and integrated using the vector field defined in the Clifford Algebra by the Clifford Neural Network $\boldsymbol{F}$.

Problem settings. We now consider a continuous gradient flow, where $\boldsymbol{x}_t$ satisfies the gradient flow equation (Eq.5). The associated infinitesimal change of variable (Eq.6) is given by [12] (Theorem.1), and the final sample probability is computed by integrating the infinitesimal change of variable (Eq.7):

$$\frac{\partial \boldsymbol{x}_t}{\partial t} = \boldsymbol{f}_t(\boldsymbol{x}_t), \quad \frac{\partial \ln p(\boldsymbol{x}_t)}{\partial t} = -\operatorname{tr}\left\{\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{x}_t}\right\}, \quad \ln p(\boldsymbol{x}_1) = \ln\ p(\boldsymbol{x}_0) - \int_0^1 \operatorname{tr}\left\{\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{x}_t}\right\} \cdots (C1).$$

When the initial samples are drawn from a given distribution $\boldsymbol{x}_0 \sim p_0(\boldsymbol{x})$, the generation process of the final samples $\boldsymbol{x}_1 \sim p_1(\boldsymbol{x})$ is called the continuous normalizing flow. CNF thus requires to compute the trace of the Jacobian, i.e. the gradient with respect to the input variable, $\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{x}_t}$. In our experiment, we use dataset consisting of M samples $\{\boldsymbol{x}_i = \{\boldsymbol{q}_i^m\}_{m=1}^N\}_{i=1}^M$ of a system of N particles ($\boldsymbol{x}_i = \{\boldsymbol{q}_i^m\}_{m=1}^N$, $\boldsymbol{q}_i^m \in \mathbb{R}^n$). We train the generative model $\boldsymbol{f}_t$ by minimizing Kullback–Leibler divergence $\mathrm{KL}(p_{t=0}(\boldsymbol{x}_0^i)|p_0(\boldsymbol{x}_0^i))$, between the prior and the sample probability given by Eq. ($C1$), where the sample $\boldsymbol{x}_0$ is computed by reverse integrating the flow $\boldsymbol{f}_t(\boldsymbol{x})$, from $t = 1$ to $t = 0$, starting from $\boldsymbol{x}_1$. The prior is an isotropic normal distribution $\mathcal{N}(0, \boldsymbol{I}_{N\times n})$ or $p_0(\boldsymbol{x}) = (2\pi)^{-(nN/2)} \exp\{-(\|\boldsymbol{x}\|^2)/2\}$. In Appendix B, we provide the details on how probability density functions are defined on Clifford Algebra and how the change of variable is implemented when using Clifford neural networks.

Results. Table 4 shows the results of DW4 and LJ13 experiments. We observe that the performance of CNF with CGGNN models is better or comparable to the other baselines. We also compare the performance of CGGNN with that of E-NF with the increased number of hidden-channel dimension, to ensure that both of the models have a comparable number of hidden units for a fair comparison. The performance of CGGNN is still comparable to or better than those baselines. These results indicate that the back-propagation through Clifford neural networks can carry informative Jacobian to transform density functions across time.

E.1 Discussion

Table 4: Comparison of the Negative Log Likelihood (and its error in terms of standard deviation) on the test partition for the considered methods, on DW4 and LJ3 dataset. The score in bold represents the best performance.

	DW4 ($n = 2$)		**LJ13** ($n = 3$)	
# training samples	10^2	10^3	10	10^2
GNF	$11.93^{\pm 0.41}$	$11.31^{\pm 0.07}$	$43.56^{\pm 0.79}$	$42.84^{\pm 0.52}$
GNF-att	$11.65^{\pm 0.39}$	$11.13^{\pm 0.38}$	$43.32^{\pm 0.2}$	$36.22^{\pm 0.34}$
GNF-att-aug	$8.81^{\pm 0.23}$	$8.31^{\pm 0.19}$	$41.09^{\pm 0.53}$	$31.5^{\pm 0.35}$
Simple-dynamics	$9.58^{\pm 0.05}$	$9.51^{\pm 0.01}$	$33.67^{\pm 0.07}$	$33.1^{\pm 0.10}$
Kernel-dynamics	$8.74^{\pm 0.02}$	$8.67^{\pm 0.01}$	$35.03^{\pm 0.48}$	$31.49^{\pm 0.06}$
E-NF	$8.31^{\pm 0.05}$	$\mathbf{8.15^{\pm 0.10}}$	$33.12^{\pm 0.85}$	$\mathbf{30.99^{\pm 0.95}}$
E-NF (24×2^n)	$\mathbf{8.24^{\pm 0.06}}$	$8.33^{\pm 0.09}$	$\mathbf{31.33^{\pm 0.30}}$	$\mathbf{30.61^{\pm 0.16}}$
CGGNN (24)	$8.80^{\pm 0.32}$	$8.56^{\pm 0.04}$	$\mathbf{31.36^{\pm 0.55}}$	$\mathbf{30.35^{\pm 0.18}}$

It is expected that CGGNN outperforms or is competitive to other baselines since the performance of the trained Clifford neural network models reported in the respective papers are better than those of baselines in their forward prediction problems. However, even taking the fact into account, we consider the performance in the inverse design and sampling with CGGNN impressive and significant. We hypothesize the reason for this is the inductive bias imposed by the geometric product of Clifford neural networks as well as the equivariance to the E(3)-group action. The key insight of the hypothesis is that the neural adjoint and CNF updates include the (discretized) integration of the functions as also in Eq. (7), whose discretization serves as physical integration methods such as the Euler integration. We claim that our hypothesis is valid, albeit indirectly and with some conditions, by giving a theoretical result that Clifford neural networks can model a quantization of the Hamiltonian equation, which means Clifford neural networks can serve as the symplectic integrator.

Proposition E.1. *Clifford neural networks serve as a quantization of Hamiltonian equation, i.e., building blocks $\boldsymbol{F} \in \mathbb{R}[X_1, X_2, \cdots, X_c]_{n,0,0}$ of Clifford neural networks have a parametrization that satisfies the following equation:*

$$(q, -p) = \boldsymbol{F}(p, q), \quad p, q \in \mathbb{R}^n.$$

This claim relies on the fact that Clifford algebras are naturally able to represent the Hamiltonian equation [23]. Based on this fact, we give a proof of Proposition E.1 in Appendix F. Besides, we conducted an ablation study in the sampling experiments, in which we compare CGGNN with its ablation without modules performing the geometric product. The results are also reported in Appendix K.1

F Connection between Clifford neural networks and Hamiltonian equations

We here claim that Clifford neural networks $\boldsymbol{F} : Cl(\mathbb{R}^n, \mathfrak{q})^{\times c_1} \to Cl(\mathbb{R}^n, \mathfrak{q})^{\times c_2}$ represent a quantization of Hamiltonian equation, relying on the fact shown in [23]:

$$x = \tilde{q} + p = p + q \otimes_{\mathfrak{q}} J, \quad J = \sum_k \boldsymbol{e}_k \otimes_{\mathfrak{q}} \tilde{\boldsymbol{e}}_k,$$

in which p, q are elements of $\mathbb{R}^n$, $\{\boldsymbol{e}_k\}$ and $\{\tilde{\boldsymbol{e}}_k\}$ are orthonormal bases of $\mathbb{R}^n$, and we work in the space of $\mathbb{R}^{2n} \cong \mathbb{R}^n \oplus \mathbb{R}^n$.

Proof of Proposition E.1. Let $\boldsymbol{x}_1 = \mathrm{inc}(p)$ and $\boldsymbol{x}_2 = \mathrm{inc}(q)$, with the following inclusion function

$$\mathrm{inc} : \mathbb{R}^n \hookrightarrow Cl(\mathbb{R}^n, \mathfrak{q}), \ \ \boldsymbol{v} \mapsto \sum_{k=1}^n \mathfrak{g}_{\mathbb{R}^n}(\boldsymbol{v}, \boldsymbol{e}_k)\boldsymbol{e}_k.$$

Then, we can build a polynomial whose computational graph is as follows:

$$\begin{tikzcd}
 & \boldsymbol{x}_1 \arrow[dr, mapsto, "\otimes_{\mathfrak{q}}"] & \\
\boldsymbol{x}_1 \arrow[ur, mapsto, "\mathrm{id}"] \arrow[dr, mapsto, "\otimes_{\mathfrak{q}}"] & & \boldsymbol{x}_1 \otimes_{\mathfrak{q}} (\boldsymbol{x}_1 \otimes_{\mathfrak{q}} \boldsymbol{x}_2) = \mathfrak{q}(\boldsymbol{x}_1)\boldsymbol{x}_2 \\
 & \boldsymbol{x}_1 \otimes_{\mathfrak{q}} \boldsymbol{x}_2 \arrow[ur, mapsto, "\otimes_{\mathfrak{q}}"] \arrow[dr, mapsto, "\otimes_{\mathfrak{q}}"] & \\
\boldsymbol{x}_2 \arrow[ur, mapsto, "\otimes_{\mathfrak{q}}"] \arrow[dr, mapsto, "\mathrm{id}"] & & \boldsymbol{x}_2 \otimes_{\mathfrak{q}} (\boldsymbol{x}_1 \otimes_{\mathfrak{q}} \boldsymbol{x}_2) = -\mathfrak{q}(\boldsymbol{x}_2)\boldsymbol{x}_1 \\
 & \boldsymbol{x}_2 \arrow[ur, mapsto, "\otimes_{\mathfrak{q}}"] & \\
Cl(\mathbb{R}^n, \mathfrak{q})^{\times 2} & Cl(\mathbb{R}^n, \mathfrak{q})^{\times 3} & Cl(\mathbb{R}^n, \mathfrak{q})^{\times 2}
\end{tikzcd}$$

When we take the geometric product with $\boldsymbol{x}_1 \otimes_{\mathfrak{q}} \boldsymbol{x}_2$ in the latter arrows, some constants can be multiplied by $\boldsymbol{x}_1 \otimes_{\mathfrak{q}} \boldsymbol{x}_2$ if necessary. □

While in appearance this diagram merely shows the channel elements $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are just swapped during the forward computation, this serves as a quantization of Hamiltonian equation, based on the fact in [23]. We presume that one reason why Clifford neural networks outperform some strong baselines in various kinds of experiences, especially those involving the prediction of future object configurations given a current state, is that building blocks of the networks can update states based on the Hamiltonian equation through the geometric product, while it also allow for the adjustment of the degree of the update by having trainable parameters in each of the arrows above.

G Experimental settings

G.1 Inverse design

To evaluate the gradient of Clifford neural networks, we adopt three types of Clifford neural networks with signatures $(3, 0, 0)$, $(3, 0, 1)$. We use as the former model a message-passing Clifford neural network (CGGNN) [50] and CGGNN with the signature $(4, 1, 0)$. We also use E(n)-equivariant graph neural network (EGNN) [53], its non-equivariant variant (GNN), and radial field network (RFN) [30] as baselines to compare. As the model with the signature $(3, 0, 1)$, we use a geometric algebra transformer [8] and as baselines we use steerable-E(3)-GNN (SEGNN) [5], geometric Clifford algebra GNN (GCA-GNN) [51], and a non-geometric transformer. We are especially interested in evaluating gradient-based optimization, also called neural adjoint method [1, 46, 60].

All the models are trained to predict bodies' positions after 1000 timesteps, where we use a similar experimental setup for the models as in [5, 8, 50, 53]. Parameters $\hat{\boldsymbol{x}}$ to be optimized are initialized by adding Gaussian noises $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$ to input data $\boldsymbol{x}$ in the test dataset. Input parameters to be optimized are initialized in the following way: For the pair $(\boldsymbol{x}, \boldsymbol{y})$ in the test dataset, we add Gaussian noises $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$ to $\boldsymbol{x}$ and define $\hat{\boldsymbol{x}} = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}$ as an initialized input parameter, and set $\boldsymbol{y}$ as the target used to define the objective function Eq.(5). We evaluate the performance of the respective models with three metrics: First metric (Obj) is the objective function. The second metric (InitDist) is MSE between $\boldsymbol{x}$ and the optimized input $\hat{\boldsymbol{x}}^*$. The third metric (TarDist) is MSE between the ground-truth target $\boldsymbol{y}$ and the output of the numerical simulator (used to generate the training dataset) for $\hat{\boldsymbol{x}}^*$. We embed the input $\hat{\boldsymbol{x}}$ to $Cl(\mathbb{R}^3, \mathfrak{q})$ to get Clifford representation when performing inference and inverse design for this input. We use Adam optimizer to optimize $\hat{\boldsymbol{x}}^{(i)}$ for 1000 iterations to get $\hat{\boldsymbol{x}}^*$.

We trained our models including the baselines with the very similar configuration provided in [50, 53, 8, 51], and use the trained models for the inverse design task. We here give the parameter configurations for the Clifford neural network models CGGNN and GATr including the baselines, as well as the hyperparameters for training and inverse-design experiments.

Models. For the CGGNN model, we use the code released by the authors: The metric for the Clifford algebra is $(p, q, r) = (3, 0, 0)$. The charges of particles are embedded as a scalar, while coordinates and velocities are embedded as 1-th grade in the Clifford algebra. The number of message-passing layers is 3.

The models E(n)-equivariant graph neural network [53], non-equivariant GNN, and radial field network (RFN) [30] are trained using the code provided in [53]. All the models are trained using Adam optimizer with respective learning rates. The number of message-passing layers is 4.

For the GATr model, we have $(p, q, r) = (3, 0, 1)$, therefore we have the Clifford algebra of dimension $2^4 = 16$. We embed object masses as 0-th grade, object coordinates as trivectors, and velocities (like translation vectors) as bivectors. Following the configurations reported in [8]. We use 10 attention blocks, 16 multivector and 128 scalar channels, and 8 attention heads. We use multi-head attention and do not use the distance-aware attention mechanism.

For the Transformer baseline, we follow a pre-layer normalization architecture with GELU activationsin the MLP block. We use 10 attention blocks, 384 channels, and 8 attention heads.

We also train non-equivariant, GCA-GNN [51]. We use the hyperparameter setting reported by [51] for their Tetris experiment.

Finally, we train SEGNN [5]. In this case, we use the code released by the authors and the hyperparameters they recommend for (slightly different) n-body experiments. This leads to the model with 0.1 million parameters. For SEGNN, we vary the number of nearest neighbors between 3 and the number of objects in the scene (corresponding a fully connected graph.)

Training. Training of the models CGGNN, EGNN, GNN, and RFN is done following the codes provided by [50, 53]. In this experiment, we set the number of objects N to be 5. Training samples are set to be 3000 and the batch size is 100. Adam optimizer was used to minimize L2 loss function and learning rate is 0.004.

The models GATr, GCA-GNN, Transformer, and SEGNN are trained following the code provided by [8]: We train the models by minimizing a L2 loss on the final position of all objects. In this experiment, the number of objects N is set to be 4. We use 10,000 steps with the Adam optimizer, using a batch size of 64 and exponentially decaying the learning rate from $3 \cdot 10^{-4}$ to $3 \cdot 10^{-6}$. Both of the training are performed using an NVIDIA A40 48 GB GPU.

Inverse optimization. For both of the coordinate and velocity optimization problems, the objective function was defined as an MSE loss function between the ground-truth future state and the output of the neural network models. We use AdamW [37] as our optimizer, and perform 1000 iterations for the optimization.

G.2 Continuous Normalizaing flow

DW-4 The Double Well experiments follow from [30, 52], where data is generated by sampling the particles' potential function

$$U_{\text{DW}}(x) = \frac{1}{2T} \sum_{ij} \left(a(d_{ij} - d_0) + b(d_{ij} - d_0)^2 + c(d_{ij} - d_0)^4 \right) \tag{16}$$

Table 5: Hyperparameters used for CGGNN model architecture, training for the model, and its inverse-design experiments

Hyperparameter name	Inverse design experiments
Hyperparameters for CGGNN architecture:	
$\Delta_{p,q,r}$	$\mathrm{diag}(1,1,1)$
Dimension of input features	3
Hidden dimension	24
Dimension of output features	1
Dimension of edge features	1
Message passing layers	3
Use residual	True
MLP for nodes: hidden layers	2
Hyperparameters for training:	
Loss function	MSE
Batch size	100
Optimizer	Adam
Learning rate	0.004
Weight decay	0.0001
Number of training samples	3000
Max training steps	100000
Scheduler	Cosine annealing
Hyperparameters for inverse design:	
Optimizer	Adam with decoupled weight decay [37]
Mean of Gaussian noise	0
Range of standard deviation to generate Gaussian noise	$[0.01, 2.0]$
Gradient clipping	False
Batch size for inverse optimization	128
Iteration number of inverse optimization	1000

with $d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ the distance between pairs of particles, and potential parameters a, b, c. T is the simulation temperature. Samples and configuration of the potential used are the ones from [30].

LJ-13 For the Lennard-Jones experiment we use the dataset generated in [30], where $M = 13$ particles are simulated and sampled whose potential energy is given by

$$U_{\mathrm{LJ}}(x) = \frac{\epsilon}{2T}\left[\sum_{ij}\left(\left(\frac{r_m}{d_{ij}}\right)^{12} - 2\left(\frac{r_m}{d_{ij}}\right)^{6}\right)\right] \tag{17}$$

with $d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, T the simulation temperature and r_m, ϵ parameters governing the shape of the potential.

Hyper-parameters and computational resources For the baselines, we consider the configuration in [52], and the hyper-partners are reported in Table 6. As the computational resource for the training, we use Vega 20 Radeon Pro, 16Gb RAM x4, and NVIDIA A40 48 GB GPU. In terms of the execution time, we also observed as already noted in [52] that computation time varies significantly with respect to the NLL, for example, both for large and small values the adaptive ODE solver may require a large number of steps, leading to long computation times. Long computation time has also the side effect of requiring increasing and unpredictable memory usage. While we could reduce and make more predictable the computation time and memory usage by fixing the number of step, we notice that 1) the computation time would still be large even when the adaptive ODE solver would require fewer steps, 2) the performance floors to a higher NLL.

H Limitations of the gradient of Clifford neural networks

The use of the gradient of Clifford neural networks has its benefits and downsides. The benefit is that due to the inductive bias of the geometric product, the gradient can have less noise (thanks to

Table 6: Hyperparameters used for Clifford neural network model architecture, training for the model, and its sampling CNF experiments

Hyperparameter name	Normalizing Flow experiments
Hyperparameters for model architecture:	
$\Delta_{p,q,r}$	$\mathrm{diag}(1, 1)$
Dimension of input features (n)	2
Hidden dimension	24
Dimension of output features	1
Dimension of edge features	1
Message passing layers	3
Use residual	True
MLP for nodes and edges: hidden layers	2
Hyperparameters for training:	
Loss function	MSE
Batch size	100
Optimizer	AdamW [36]
Learning rate	0.004
Weight decay	0.0001
Number of training samples	$10, 10^2, 10^3$ (see experiments)
Max training steps	1'000 (1000, 300 for DW4, 500, 1000 for LJ13)
Scheduler	None
ODE method	dopri5 (Runge-Kutta 4(5) of Dormand-Prince) [58]
relative tolerance	10^{-4}
absolute tolerance	10^{-4}
Error	we performed the simulation using either random seeds or consecutive seeds. We early discarded simulation with too large validation NLL that leads to large computation time for the ODE solver, with at least 3 repetitions.
Hyperparameters for Validation and Test:	
Batch size	100
Number of samples	1000 (validation and test, the first 2'000 samplers of the dataset)
ODE method	dopri5 (Runge-Kutta 4(5) of Dormand-Prince) [58]
relative tolerance	10^{-4}
absolute tolerance	10^{-4}
DW Dataset:	
Number of particles (N)	4
dimension (n)	2
a parameter	0.9
b parameter	-4
c parameter	0
d_0 distance offset	4
LJ Dataset:	
Number of particles	13
dimension	3

the geometric bias) and is directly passed through the input compared to that of neural networks that work on Euclidean spaces which need to learn the bias by making it have potentially redundant parameters. The limitation is the computational complexity of Clifford neural networks: In the experiments, we experienced up to 20$\times$ slower computational speed than the baselines. The cause of this complexity mostly comes from the implementation of geometric product, which is performed on all the pairs of basis elements composing multi-vectors. Also in the experiment of Section E, due to the expressiveness of Clifford neural networks, the adaptive ODE solvers such as dopri5 took long to solve ODEs integration. We actually used existing datasets, so we inherited the limitations of those datasets and baselines. In particular, we are not considering out-of-distribution samples, since these analyses are out of the scope of the current work. However, we emphasize that our implementation is quite straightforward in the sense that we simply use Clifford neural networks with initial settings with few parameter changes or replace some models with Clifford neural networks, and this fact can allow future work to transcend these limitations: For the former issue, while the introduction of Clifford Algebra increases the computational complexity, the scaling factor of the synthetic complexity does not change, for example in terms of the number of samples or number of particles. We might be able to reimplement the models with JAX software [4]. The second issue would be also manageble, since the problem is mostly stemmed from the process of Neural ODEs and some of the works in Neural ODEs proposed efficient architectures [21, 15].

I Rutime comparison results

Table 7: Ratio of the computation time of analytic gradient to that of autograd gradients for different batch and channel sizes and for different signatures.

signature	k	(1, 1)	(1, 10)	(100, 1)	(100, 10)
(2, 0, 0)	2	2.08	2.29	2.04	1.20
(2, 0, 0)	3	2.55	2.69	2.45	1.36
(2, 0, 0)	4	2.59	2.97	2.70	1.38
(2, 0, 0)	5	2.92	3.10	2.86	1.32
(2, 0, 1)	2	4.89	3.45	2.21	1.12
(2, 0, 1)	3	4.23	4.30	2.52	1.15
(2, 0, 1)	4	4.80	4.59	2.68	1.13
(2, 0, 1)	5	5.02	4.78	2.68	1.13
(3, 0, 0)	2	3.34	3.47	2.28	1.09
(3, 0, 0)	3	4.25	4.23	2.55	1.08
(3, 0, 0)	4	4.57	4.60	2.65	1.09
(3, 0, 0)	5	4.90	4.73	2.80	1.08
(3, 0, 1)	2	0.75	3.04	3.70	1.11
(3, 0, 1)	3	6.00	5.85	4.29	1.05
(3, 0, 1)	4	8.73	6.48	4.68	1.01
(3, 0, 1)	5	8.36	6.85	4.97	1.03
(3, 1, 0)	2	6.50	5.70	3.55	1.06
(3, 1, 0)	3	8.10	6.47	4.52	0.98
(3, 1, 0)	4	8.68	5.46	4.59	0.99
(3, 1, 0)	5	8.99	5.69	4.88	0.96
(4, 1, 0)	2	11.02	8.40	1.84	0.50
(4, 1, 0)	3	13.49	9.18	2.13	0.45
(4, 1, 0)	4	15.54	10.46	2.07	0.42
(4, 1, 0)	5	16.62	10.41	2.05	0.44

Table 8: Absolute computation times for analytic gradients for different batch, channel sizes, and for different signatures. Computation time for 100 repetitions and normalized by batch and channel size. The time rounds to zero when the time is below 0.005 seconds. Computation time for different signatures seems to depend only on the total size of the algebra and not on the signature itself. Computation time increases with polynomial order as expected, since multiple geometric product operations are necessary.

signature	k	(1, 1)	(1, 10)	(100, 1)	(100, 10)
(2, 0, 0)	2	0.42	0.04	0.00	0.00
(2, 0, 0)	3	0.75	0.09	0.01	0.00
(2, 0, 0)	4	1.03	0.11	0.01	0.00
(2, 0, 0)	5	1.18	0.13	0.01	0.00
(2, 0, 1)	2	0.68	0.06	0.01	0.00
(2, 0, 1)	3	0.94	0.12	0.01	0.00
(2, 0, 1)	4	1.41	0.16	0.02	0.00
(2, 0, 1)	5	1.83	0.21	0.02	0.00
(3, 0, 0)	2	0.46	0.05	0.01	0.00
(3, 0, 0)	3	0.88	0.10	0.01	0.00
(3, 0, 0)	4	1.29	0.14	0.02	0.00
(3, 0, 0)	5	1.71	0.19	0.02	0.00
(3, 0, 1)	2	1.05	0.11	0.01	0.00
(3, 0, 1)	3	1.76	0.22	0.03	0.00
(3, 0, 1)	4	2.59	0.33	0.04	0.01
(3, 0, 1)	5	3.43	0.41	0.05	0.01
(3, 1, 0)	2	1.01	0.11	0.02	0.00
(3, 1, 0)	3	1.90	0.21	0.03	0.00
(3, 1, 0)	4	2.79	0.29	0.04	0.01
(3, 1, 0)	5	3.68	0.38	0.05	0.01
(4, 1, 0)	2	2.16	0.26	0.03	0.01
(4, 1, 0)	3	4.24	0.50	0.06	0.02
(4, 1, 0)	4	6.34	0.76	0.10	0.02
(4, 1, 0)	5	8.33	0.99	0.12	0.03

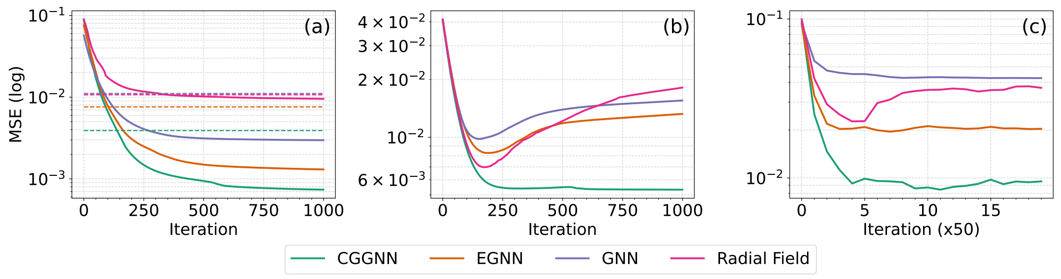

Figure 5: **Numerical results of inverse optimization for coordinate of neural network model f_θ.** (a) shows optimization curve (Eq. 5) for inverse-design at different iteration steps. The horizontal dotted lines are the final test errors observed in the training phase of respective models, which are comparable to the test errors reported in [50]. (b) is the MSE error between ground-truth input and optimized input, and (c) shows MSE error between the ground-truth target state and simulated target with respect to the optimized coordinate input. All the errors are averaged over the simulation results of 128 trajectories.

Table 9: **Comparison of the inverse design results with baselines.** All the scores in the table is reported in MSE ($\times 10^{-3}$).

	Coordinate			**Velocity**		
	Obj	InitDist	TarDist	Obj	InitDist	TarDist
RFN	9.54	18.17	36.92	4.19	92.79	30.23
GNN	2.98	15.55	42.49	5.53	121.70	87.67
EGNN	1.31	13.23	20.38	**2.50**	88.65	66.24
CGGNN	**0.74**	**5.33**	**9.52**	2.81	**56.64**	**5.72**

J Additional results on inverse design

J.1 Results of inverse design with CGGNN for coordinate

Here, we discuss the design results of coordinate with CGGNN and baselines, as well as those of velocity discussed in Section 4. The result is shown in Figure 5. We observe that CGGNN outperforms all of the baselines. Notably, while the objective of all the models keeps decreasing during the optimization, the distance between the inputs of the models (Figure 5 (b)) keeps increasing after around 200 iterations, except CGGNN. Taking into account the observation that the distance between the ground-truth target and simulated targets based on the optimized input remain lower than that of the other models, CGGNN is robust to noises added to the input, while other baselines are more likely to fall into local optimal.

J.2 Results of Geometric Algebra Transformer

Experimental setting..

For the inverse design, we use the same experimental setting described in Section 4, which we also give an explanation in Appendix G.1.

We here report the results of the inverse design with the geometric algebra transformer (GATr) [8]. The geometric transformer is categorized as a model with the signature $(3, 0, 1)$. We train GATr, as well as some baselines steerable-E(3)-GNN (SEGNN) [5], geometric Clifford algebra GNN (GCA-GNN) [51], and a non-geometric transformer, using the same configuration provided in [8].

Results. Table 10 shows the inverse desing result of GATr and its baselines. We observed that two Clifford algebra-based models, GATr and GCA-GNN, outperform the other models in most of the cases. This indicates, albeit empirically, that while we do not know the theoretical guarantee of Corollary 3.1 for the signature with $r > 0$ case, the arithmetic carried out by Clifford neural

Table 10: **Comparison of the inverse design results with baselines for coordinate input.** The score reporeted is MSE ($\times 10^{-4}$). The standard deviations are 0.1, 0.2, 0.4 and 0.8.

	0.1		0.2		0.4		0.8	
	Obj	InitDist	Obj	InitDist	Obj	InitDist	Obj	InitDist
Transformer	0.17	14.49	2.63	53.61	7.36	82.27	17.64	118.45
SE-GNN	0.17	16.18	7.67	**22.75**	87.20	147.93	32.61	87.10
GCA-GNN	0.19	51.64	**1.35**	50.73	**6.25**	**58.12**	19.17	**70.83**
GATr	**0.13**	**1.11**	2.14	22.97	7.59	128.29	**16.16**	549.35

networks pass informative gradient through the input of the networks. We also observed that while GATr performed well for the case of a small standard deviation, the performance of GCA-GNN was relatively stable compared to the other baselines across different magnitudes of the standard deviation.

J.3 Robustness assessment in inverse design

We here show the additional results of the inverse design of coordinates and velocity, by changing the standard deviation of Gaussian distribution from which we sample noises. In both of the cases, CGGNN basically outperforms all of the models. We emphasize that the aim of this objective is to make sure that gradient of Clifford neural network produces reasonable results, given the prediction performance reported in the literature.

J.4 Inverse design of coordinates of the objects

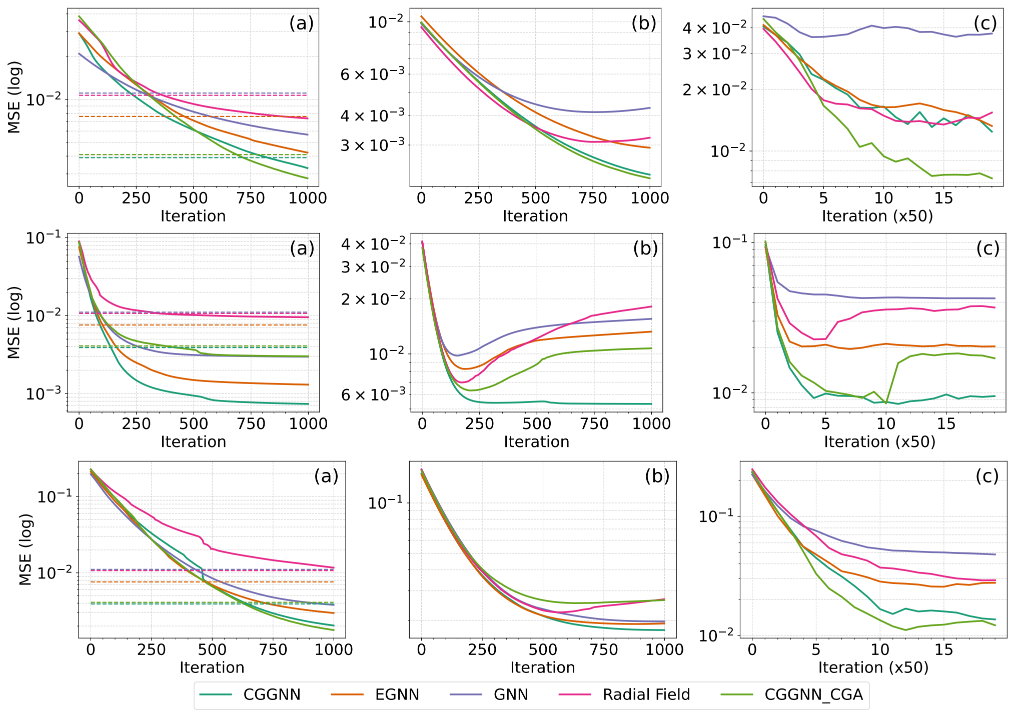

Figure 6: **Numerical results of inverse optimization for coordinate of neural network model f_θ with different standard deviations**. (a), (b), and (c) are the same as described in Figure 5. Each row corresponds to the inverse design results with different standard deviations: 0.1, 0.2, and 0.4.

J.5 Inverse design of velocity of the objects

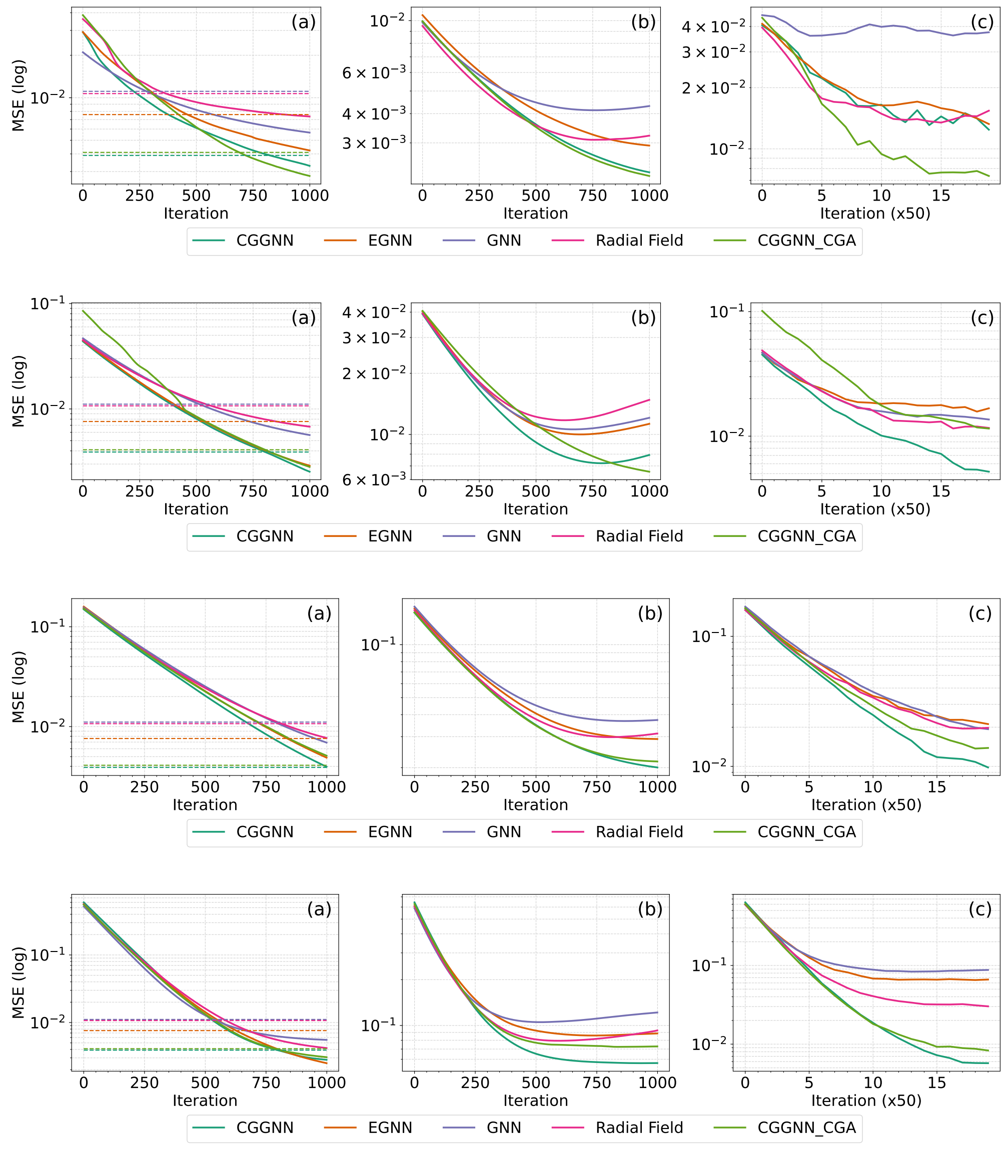

Figure 7: **Numerical results of inverse optimization for coordinate of neural network model f_θ with different standard deviations**. (a), (b), and (c) are the same as described in Figure 5. Each row corresponds to the inverse design results with different standard deviations: 0.1, 0.2, 0.4., and 0.8.

K Additional results on sampling from probability distributions

K.1 Ablation study

We performed an ablation study on the importance of the use of the geometric product in the Clifford neural network. We compare CNF with CGGNN with that of CGGNN without the geometric product layer. The results are shown in Table 11. We observed that substantial difference in the peformance between the CNF with CGGNN and its ablation model in all the experiments. This indicates the CGGNN benefits largely from the geometric product layers.

Table 11: **Ablation study for the role of geometric product in the Clifford Neural Network.** The score is reported as Negative Log-Likelihood (NLL) and its standard deviation, while the number of features is denoted beside the model name.

	DW4 ($n = 2$)		**LJ13** ($n = 3$)	
# training samples	10^2	10^3	10	10^2
CGGNN (24)	$8.80^{\pm 0.32}$	$8.56^{\pm 0.04}$	$31.36^{\pm 0.55}$	$30.35^{\pm 0.18}$
CGGNN w/o GP (24)	$10.99^{\pm 0.35}$	$11.44^{\pm 0.79}$	$38.40^{\pm 1.91}$	$39.13^{\pm 0.99}$

L Related work

Clifford neural networks. Clifford neural networks are an emerging class of geometric deep learning models [9]. All the representation and arithmetic inside the networks are based on Clifford algebra: Physical features of the systems are represented by multi-vectors, and the inherent interaction of physical features are modeled by the geometric product and linear summation of multi-vectors. Various kinds of machine learning models incorporate Clifford algebra and employ arithmetic as their inductive biases. Such examples include Fourier neural networks, [6], transformers [8], geometric message passing neural networks [33, 50, 54], image-based convolutional neural networks [61]. We can easily introduce geometric inductive biases when we use of Clifford algebra in neural networks using the geometric product. The geometric product can model various kinds of physical interactions, such as the interaction between the pressure and velocity fields in fluid dynamics simulations in a data efficient way. The effectiveness of having those biases is shown in diverse applications in the literature from simple regression tasks such as volume prediction to challenging tasks such as solving fluid dynamics simulations.

Inverse Design. Inverse problems, such as inverse optimization of system parameters and inverse parameter inference, are one of the important classes of problems in both of scientific and engineering domains. Such problems have a huge importance in engineering, *e.g.* in designing jet engines [2] and materials [10] where the objective can be minimizing drag or maximizing durability, and inverse parameter inference (*i.e.* history matching) [57, 59, 43] where the objective can be maximum a posteriori estimation. To solve such problem, classical methods include adjoint method [55, 56], shooting method [26], collocation method [3], etc. One recent work [1] explores optimization via backpropagation through differential physics in the input space, demonstrating speed-up and improved accuracy compared to classical CEM method [48]. The method is categorized as the neural adjoint method [1, 60, 46] and shown to be computationally efficient and scalable to high-dimensional input space compared to CEM method.

Neural adjoint method. This method estimates the gradient of the objective function defined by loss function $\mathcal{L}$ and neural network simulator $\boldsymbol{f}_\theta$ with trainable parameters θ to perform iterative gradient-based estimation procedure:

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{f}_\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y})}{\partial \boldsymbol{x}}(\boldsymbol{x}^{(i)}).$$

Here, i represents an iteration number, y is the target output, and α is an optimization rate. This update rule is well-defined as long as the neural network $\boldsymbol{f}_\theta$ is differentiable with respect to its input $\boldsymbol{x}$. We note that while optimizing the parameter $\boldsymbol{x}$, we fix the model parameter θ. One advantage of the neural adjoint method is its task-agnostic generalization capability – Neural networks are trained on dynamics with no access to the task objective or design space and can be used to solve new design tasks, as opposed to the adjoint method based on classical solvers that typically cannot generalize to tasks or designs outside of distribution.

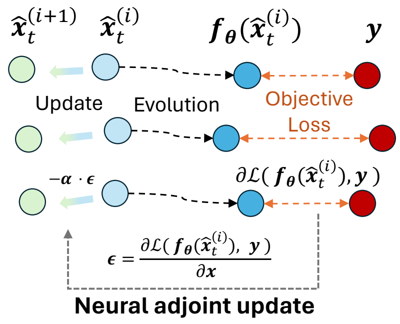
$\hat{\boldsymbol{x}}_t^{(i+1)}$
$\hat{\boldsymbol{x}}_t^{(i)}$
$\boldsymbol{f}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t^{(i)})$
$\boldsymbol{y}$
Update
Evolution
Objective Loss
$-\boldsymbol{\alpha} \cdot \boldsymbol{\epsilon}$
$\partial\mathcal{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t^{(i)}), \boldsymbol{y})$
$\boldsymbol{\epsilon} = \frac{\partial\mathcal{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t^{(i)}), \boldsymbol{y})}{\partial \boldsymbol{x}}$
Neural adjoint update

Figure 8: **Schematic of Neural Adjoint method**. Parameter $\hat{\boldsymbol{x}}_t^{(i)}$ is updated by back-propagating gradient of $\mathcal{L}(\boldsymbol{f}_\theta(\boldsymbol{x}), \boldsymbol{y})$, with the target position $\boldsymbol{y}$.

Normalizing Flows. Normalizing flows form one large class of generative modeling frameworks. Fundamental work is [47], which proposes to transform a simple base distribution, typically Gaussian distribution, into a target distribution by applying a series of bijective mappings, each parameterized by neural networks. Several variations of normalizing flows have been proposed to enhance their flexibility and expressiveness. Examples of the artchitectures are [27, 13, 16, 30]. The normalizing flows have been also applied to various kinds of applications such as molecular generations: Boltzman generator [41], equivariant normalizing flows [53].

Neural ODEs. Neural Ordinary Differential Equations (Neural ODEs) offer a framework for continuous-time modeling in neural networks [12]. Neural ODEs reformulate the discrete transformations of ResNet-like architectures into continuous-time transformations and model the evolution of hidden states using a differential equation parameterized by a neural network, enabling adaptive computation and potentially infinite depth. Recent advancements have focused on improving the stability and expressiveness of Neural ODEs. Such examples include memory-efficient Neural ODEs [21], and [15] which introduce augmented states to enhance representational capacity developed Latent ODEs for learning from latent variable models. As also explained in the section on normalizing flow, neural ODEs are also applied in other frameworks such as normalizing flows [13, 16, 27, 30] and flow-matching [11, 29].

Gradient in machine learning . Representative usage of the gradient of neural network models is to update parameters of the models so that the likelihood for the models to fit given data is maximized [32]. Therein, the gradient used to optimize models is seen as being defined on the *parameter space* of the models, in which model parameters are considered as the input of loss function while fixing the input and output of the models with observed data during the training. Clifford neural networks proposed in the recent literature [6, 8, 33, 39, 50, 54, 61] also make this assumption when updating their model parameters. Another emerging approach is to compute the gradient with respect to the *input space* of neural network functions. A common underlying assumption in this approach and its application is that functions are defined on Euclidean space or smooth manifolds. Thereon, the notion of differentiability of functions is well-defined, and therefore the gradient of these functions can be passed through their input. Applications of the gradient in this regard can be found in image denoising [35, 49], adversarial example generation [22, 31, 40], and feature visualization [42, 17]. In the context of scientific problems, we can also find inverse design [1, 60], flow-matching [11, 29], and normalizing flow [13, 16, 27, 30].

Untriviality of taking the gradient of Clifford neural networks. Clifford neural networks have been typically applied for tasks that do not require computing the gradient of these networks with respect to their input variables, despite the existence of automatic differentiation tools such as Autograd [44]. We believe this limitation arises because the concept of differentiability in Clifford neural networks is non-trivial. This non-triviality stems from the fact that Clifford algebra is inherently algebraic and does not incorporate a metric or distance structure, which is crucial for defining differentiability. Even in a recent study [61], in which Clifford-based convolutional neural networks are proposed using the notion of pseudo-Riemannian manifolds, differentiability between the manifolds is not addressed. Thus the study on the differentiability of functions between Clifford algebras remains an underexplored and not yet fully understood area. We extend the notion of differentiability introduced by [18, 19, 20] and show the gradient of Clifford neural networks is compatible with that of functions on Euclidean space. These results eventually ensure the validity of the usage of automatic differentiation modules such as Autograd [44] for Clifford neural networks.

M Broader social impact

This section discusses the broader social impact of the presented work. Our work has important implications for the physical systems and engineering, as many problems in these fields require multibody simulations; we also discuss it in Section 1 and 2. Although this work focuses on standard benchmark tasks, our experiments demonstrate the validity of the usage of our theoretical results for important classes of physical systems. Beyond simply applying our idea to standard benchmark tasks, the idea can be applied to tasks in real-world applications, such as molecular generation, designing boundaries in fluid, and many more.

Our work has no obvious negative social impact. As long as it is applied to the physical sciences and engineering in a way that benefits society, it will have positive effects.

N Asset licenses

The following resources and assets have been used (directly or indirectly):

- DW-4 & LJ-4 datasets and code `https://github.com/noegroup/bgflow`: MIT license
- EGNN model and code (`https://github.com/vgsatorras/egnn`): MIT license
- E-NF model, data and code `https://github.com/vgsatorras/en_flows`: MIT license
- FFJORD solver and code (https://github.com/rtqichen/ffjord): MIT license
- GATr model and code (`https://github.com/Qualcomm-AI-research/geometric-algebra-transformer`): BSD-3-Clause-Clear license

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   We propose to bridge the gap in the understanding of the abstract concept of the gradient of functions defined as transformations between Clifford Algebras and the practical use of Autograd in modern frameworks such as PyTorch [45] or JAX [4] (Section 3, "Differentiable function on Clifford algebra" and "Connection on gradient between base space and associated Clifford algebra.", and Appendix A).

   We formally introduce the definition of gradient via the directional gradient (Section 3) and then provide the connection with the Euclidean metric (Section 3).

   Further, we provide two classes of experiments: inverse design (Section 4) and sampling from a probability distributions (Section E) to justify the theoretical and practical use of Clifford gradients.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitation in Appendix H. We raise a high computational complexity of Clifford neural networks as the limitation in the current experiments.

   While the introduction of Clifford Algebra increases the computational complexity, the scaling factor of the synthetic complexity does not change, for example in terms of the number of samples or number of particles.

   On the other side, we removed previous limitations in the definition of gradients to only the signature of Clifford Algebra of the type $(n, 0, 0)$, and we extended the definition to the general case (p, q, r) (section 3).

   On the side of the experiments, we used existing datasets, so we inherited the limitations of those datasets and baselines. In particular, we are not considering out-of-distribution samples, since these analyses are out of the scope of the current work.

   We evaluate in Appendix J and Appendix K the robustness of the results in function for example of the deviation of the parameters with respect to the initial configuration, or the ablation study of the geometric product.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We provide a comprehensive proof for Proposition C.1 in Appendix C. We also give a concise proof for Corollary 3.1 in the main text. The proof for Proposition E.1 can be found in Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in the appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification:

   While we provide a demo implementation in `https://anonymous.4open.science/r/clifford_flow_private-EC9C` and the final code will be released at acceptance, we describe in both main text sections 4, E,and Appendix G, the experimental problems, the datasets, and configurations.

   Although it is not always possible to capture all the information, we provide an extensive description of the hyper-parameters used in the two Experiments. In the final code release, the code includes a script to reproduce the results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case

of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide a demo implementation in `https://anonymous.4open.science/r/clifford_flow_private-EC9C` with the data necessary for the two experiments of section E, while the final code will be released at acceptance, We describe in both main text sections 4, E,and Appendix G, the experimental settings, the datasets, and configurations.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

Justification: We explain the experimental settings in Section 4 and Section E. Details including hyperparameter configuration for the experiments are included in Appendix G.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in the Appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide the experimental results, with error intervals in terms of standard deviation, in Table 4, Table 9, Table 10, Table 11, where data is divided in train, validation (when needed) and test. Additional information is reported in Appendix G.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The information on computational resources for the experiments is provided in Appendix G, for both inverse design (Appendix G.1) and sampling from distributions (Appendix G.2).

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The presented work follows the NeurIPS Code of Ethics. We also include Appendix M, which discusses the broader social impact.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss social impacts on the present work in Appendix M.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification:

    The paper poses no such risks since no language model, neither image generation, nor scraped datasets are used.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited, and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We referred to the data, code, and models and provided references to the papers of the original authors, in the main text and appendix. Also in the code, we mention the most necessary libraries and projects. We list the licenses of the resources used in Appendix N.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We give an explanation on our idea and its associated experiments in the main text and appendix. The instruction for the experiments is also included in `https://anonymous.4open.science/r/clifford_flow_private-EC9C`. But, we used dataset and codes from publicly available projects. Existing assets are listed in Appendix N.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

    The paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

Justification:
The paper does not involve crowdsourcing nor research with human subjects.