Orthogonium : A Unified, Efficient Library of Orthogonal and 1-Lipschitz Building Blocks

Thibaut Boissin [1 2 3] **Franck Mamalet** [1 2] **Valentin Lafargue** [1 2 4] **Mathieu Serrurier** [3 2]

Abstract

Orthogonal and 1-Lipschitz neural network layers are essential building blocks in robust deep learning architectures, crucial for certified adversarial robustness, stable generative models, and reliable recurrent networks. Despite significant advancements, existing implementations remain fragmented, limited, and computationally demanding. To address these issues, we introduce **Orthogonium** , a unified, efficient, and comprehensive PyTorch library providing orthogonal and 1-Lipschitz layers. Orthogonium provides access to standard convolution features—including support for strides, dilation, grouping, and transposed-while maintaining strict mathematical guarantees. Its optimized implementations reduce overhead on large scale benchmarks such as ImageNet. Moreover, rigorous testing within the library has uncovered critical errors in existing implementations, emphasizing the importance of standardized and reliable tools. Orthogonium thus significantly lowers adoption barriers, enabling scalable experimentation and integration across diverse applications requiring orthogonality and robust Lipschitz constraints. Orthogonium is available here.

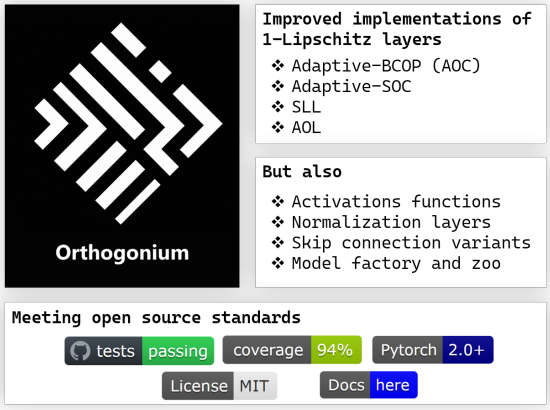

Figure 1: Orthogonium offers a standardized API to use, and create 1-Lipschitz layers, allowing a user to construct, test, and improve easily such a kind of network

1. Introduction

1-Lipschitz neural networks constrain transformations to preserve input norms, providing tight, certifiable robustness against adversarial attacks (Szegedy et al., 2014; Anil et al., 2019). Orthogonal layers reinforce the 1-Lipschitz constraint by requiring an exact unity constant in almost all directions, providing tighter global certification guarantees. Besides robustness, these layers benefit normalizing flows (Kingma & Dhariwal, 2018; Behrmann et al., 2019), Wasserstein GANs (Arjovsky et al., 2017; Gulrajani et al., 2017), stable recurrent architectures (Kiani et al., 2022; Qi et al., 2020; Bansal et al., 2018), and physics-informed models.

Motivations. Over the last decade, the certifiable-robustness community has produced an impressive toolbox of 1-Lipschitz building blocks—orthogonal and norm-controlled layers, specialized activations, residual schemes, and normalization layers. Unfortunately, these ingredients remain scattered across dozens of papers and repositories. For practitioners who simply "need a 1-Lipschitz backbone"—be they working on Wasserstein GANs, stable RNNs, privacy-preserving analytics, or safety-critical perception—the landscape is opaque: Which methods are still maintained? Which supports modern CNN staples such as stride, dilation, grouping, or transposed convolutions? Which combinations scale to ImageNet?

Constructing a truly 1-Lipschitz network exacerbates the problem. Every layer in every branch must respect the

[1]Institut de Recherche Technologique Saint-Exupéry, Toulouse, France [2]Artificial and Natural Intelligence Toulouse Institute, France [3]IRIT, Toulouse, France [4]now at IMT, Toulouse, and INRIA, Bordeaux, France. Correspondence to: Thibaut Boissin <thibaut.boissin@irt-saintexupery.com>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

global constraint, yet many recent proposals cover only the vanilla 3×3 convolution and ignore grouped, dilated, or strided variants that dominate contemporary architectures (Liu et al., 2022; Tan & Le, 2019; Sandler et al., 2018; Ronneberger et al., 2015). In practice, researchers resort to copy-pasting the original authors' code—sometimes years out of date—because re-implementing and validating the underlying mathematics (let alone optimizing kernels) is prohibitively time-consuming. Convolutional layers are a prime example: half a dozen orthogonalization schemes exist, but none has achieved field-wide consensus, and their relative merits are hard to benchmark because no common interface or test-bed exists.

Training itself is also expensive. Certifiable objectives typically require longer schedules to converge, and per-batch cost grows as soon as weights are iteratively projected or parameter matrices inflated for numerical stability. Here, implementation details matter: an efficient CUDA kernel or memory-lean fusion can translate directly into larger batches, deeper models, or simply more optimization steps on the same hardware budget.

These main points motivate **Orthogonium** . By *centralizing* published methods behind a unified PyTorch API, *standardizing* their signatures and test coverage, and prioritizing *efficient, scalable* kernels, the library (i) turns method comparisons into one-liner swaps, (ii) lowers the entry-barriers for neighboring fields to adopt 1-Lipschitz layers, and (iii) makes large-scale experiments—ImageNet-1K or semantic segmentation tasks—practically feasible.

Our Contributions. To address these challenges, we introduce **Orthogonium** , a unified, efficient library that combines theoretical rigor and practical implementation. Our main contributions are:

- **Unified, Explicit API:** A comprehensive, PyTorch-friendly implementation covering dense, convolutional, and hybrid orthogonal layers, explicitly constructed in the spatial domain for straightforward integration.
- **Full Feature Parity:** Native support for essential convolutional operations—striding, dilation, transposition, and grouping—allowing seamless integration into modern network architectures.
- **Efficient and Scalable Implementation:** Optimized kernels provide high performance, with minimal overhead (approximately 10% slowdown) compared to unconstrained convolutions on large-scale benchmarks like ImageNet (Deng et al., 2009).
- **Cross-Fertilization and Flexibility:** Orthogonium provides modularity to swiftly explore hybrid approaches, enhancing existing methods such as SOC, SLL, and Sandwich layers, promoting broader adoption.
- **Extensive Validation and Testing:** Our rigorous unit testing identified and corrected subtle implementation errors in published repositories, improving reliability and correctness across all supported methods.

By unifying orthogonality, flexibility, and computational efficiency, Orthogonium represents a significant advancement, enabling researchers and practitioners to integrate orthogonal layers seamlessly into a wide variety of deep learning applications. The remainder of the paper is structured as follows: Section 2 introduces our dense layer implementations, Section 3 covers orthogonal convolutions, and Section 4 covers the approach we used to unit test all our layers. The issues identified in certain approaches underscore the importance of open source tools for safety-critical applications.

2. Dense Orthogonal Layers

Orthogonium provides an efficient and flexible implementation of orthogonal dense layers with a straightforward, drop-in PyTorch interface, `OrthoLinear`, supporting several orthogonalization methods. This approach simplifies integration into existing workflows and allows users to choose methods suited to their computational constraints and stability requirements. Below, we outline the supported methods and their characteristics:

Unified API with `OrthoLinear`. The provided `OrthoLinear` class extends `torch.nn.Linear`, ensuring seamless integration into standard PyTorch models. It enforces orthogonality constraints through parameterizations registered via the customizable `OrthoParams` object, which encapsulates both spectral normalization and orthogonalization methods.

Supported Orthogonalization Methods. Orthogonium supports five distinct orthogonalization algorithms, each appropriate for different scenarios: Björck–Bowie Iterative Projection, Exponential Map method, Modified Gram–Schmidt (QR Decomposition), Cayley Transform, Cholesky Decomposition. These parametrizations are fully compatible with the PyTorch `parametrize` API.

Spectral Normalization. To ensure numerical stability and enforce Lipschitz constraints, spectral normalization via batched power iteration is applied optionally before orthogonalization. This preconditioning enhances the stability and convergence of the orthogonalization processes. Spectral normalization can also be used in a standalone way, leading to 1-Lipschitz layers.

Flexibility and Extensibility. Users can easily customize orthogonalization and normalization methods through the `OrthoParams` object. Orthogonium provides several predefined configurations depending on the desired method.

Implementation Efficiency. Implementation and efficiency are crucial factors in the selection of a layer. This is why Orthogonium provides layers with some non-trivial modifications in order to be more scalable.

By providing a unified API and efficient implementations, Orthogonium 's dense orthogonal layers enable easy integration, rigorous validation, and high-performance execution in diverse deep learning applications.

3. Orthogonal Convolutions

Orthogonium implements multiple classes of 1-Lipschitz and orthogonality-preserving convolutions—allowing for a user to choose between exactness, speed, and architectural flexibility— plus two 1-Lipschitz extensions that embed the convolutions inside higher-order residual blocks. Table 2 summarizes their properties, while the paragraphs below describe design choices and implementation tweaks; algorithmic derivations are deferred to Appendix B. As the exposed layers can differ significantly from their original implementations, original layers are available in the `legacy` module.

Adaptive Orthogonal Convolution (AOC). AOC is the default constructor, `AdaptiveOrthoConv2d`/`ConvTranspose2d`, and generalizes BCOP kernels (Li et al., 2019) to any kernel size, stride, dilation, groups natively (i.e, without resorting to reshaping tricks or FFTs). Transposed convolutions are also supported natively. It is based on the method described in (Boissin et al., 2025). The layer materializes an explicit weight tensor whose forward path is a single call to `torch.nn.Conv2d` This approach yields a $\leq 1.13\times$ wall-time over plain `Conv2d` on ImageNet-1k at batch size 256.

Adaptive-SOC. Skew Orthogonal Convolution (SOC) offers orthogonal training by parameterizing the kernel as the exponential of a skew-symmetric filter (Singla & Feizi, 2021b). Orthogonium 's `AdaptiveSOCConv2d`/`ConvTranspose2d` fuses SOC with AOC's stride-aware approach, stores the explicit exponential once per update (making its cost independent of the batch size), and supports grouped, dilated or transposed variants out-of-the-box—reducing memory (See Appendix B.2). Also, this method relies on a normalization step. We used "AOL" instead of the original "fantastic four"(Singla & Feizi, 2021a) approach, making the convergence quicker than the original method. (3-4 iterations instead of the original 6).

Almost-Orthogonal Layers (AOL). When strict orthogonality is unnecessary, `AOLConv2d` implements the almost-orthogonal method of Prach & Lampert (Prach & Lampert, 2022). The re-parametriser is registered through PyTorch's `parametrize` API and guarantees a layer Lipschitz constant ≤ 1, while remaining fast. Orthogonium implements a multi-step variant, making use of the proximity between this approach and the Gram iteration described by (Delattre et al., 2024). This variant allows a tighter normalization than the original method.

SDP-based Lipschitz Layers (SLL). (Araujo et al., 2023) defined a 1-Lipschitz residual blocks that bundle a $\sigma(\cdot)$-non-linearity inside the convolution, while offering a tight Lipschitz normalization. We extended the original `SDPBasedLipschitzConv` to support groups and dilation. We also designed its down-sampling equivalent `SLLxAOCLipschitzResBlock` which allows for stride and dimension change with an AOC kernel at their core to enable strides and channel changes, similarly as the Resnet downsampling blocks (Fig. 3); details are given in Appendix B.3.

Sandwich-AOC. Finally, Orthogonium replaces the costly frequency-domain Cayley step of "sandwich layers" (Wang & Manchester, 2023) with an explicit AOC kernel, removing complex-valued FFTs, making this layer efficient for large input images (e.g, 224×224); details are given in Appendix B.4. This layer is still under development and will be available soon.

Other Modules: Activations, Normalization, and Residual Blocks. Beyond convolutions, **Orthogonium** centralises the necessary layers needed to build fully 1-Lipschitz networks.

Activations. Five gradient-norm-preserving nonlinearities are shipped in `custom_activations`: `Abs`, `SoftHuber`, `MaxMin`, `HouseHolder`, and its second-order variant. Each is unit-Lipschitz by construction; for the Householder family we patched the missing $1/\sqrt{2}$ scale factor, restoring $\sigma_{\max}(J) = 1$ to machine precision (see Appendix C). Detailed APIs appear in the online docs.

Normalization. Instead of BatchNorm—which destroys Lipschitz control— Orthogonium offers `BatchCentering` and `LayerCentering`. Both subtract running means but leave variances untouched, so they preserve feature-map norms at inference; their implementation lives in `normalization.py`.

Since the usual residual connection is not 1-Lipschitz,

Figure 2: Implemented convolutional layers in Orthogonium . All run on GPU, accept `stride`, `dilation`, `groups`, `padding_mode`, and have (when possible) parity with `nn.Conv2d`.

Layer	Orthogonality	Key use-case	Internal method
AOC (`AdaptiveOrthoConv2d`)	exact	general CNN backbones	lifted BCOP / RKO
Adaptive-SOC	exact	depthwise / small kernel size	exponential skew filter
AOL	$\leq$ 1-Lip. ($\approx$)	fast training	multi-step projection
SLL / SLL-AOC	$\leq$ 1-Lip. (tight)	residual blocks	AOL + SDP constraint
Sandwich-AOC	$\leq$ 1-Lip. (tight)	tight Lipschitz estimation without orthogonality	AOC pair

Orthogonium provides several lightweight residual wrappers designed to combine an arbitrary internal function fn—typically an orthogonal convolution followed by non-linear activation—with a skip connection, ensuring that the resulting residual blocks remain exactly 1-Lipschitz. These wrappers, implemented as minimal 'torch.nn.Module' classes, preserve the benefits of skip connections while strictly enforcing global Lipschitz constraints.

Among the various strategies, *ConcatResidual* splits the input along the channel dimension, applies a function fn to one half, and concatenates it back with the untouched half, ensuring the resulting block remains 1-Lipschitz if fn is. *L2NormResidual* combines the identity and residual branches using an ℓ_2 average, specifically outputting $\sqrt{\frac{1}{2}x^2 + \frac{1}{2}\mathrm{fn}(x)^2 + \varepsilon}$ (with small ε for numerical stability), ensuring exact 1-Lipschitz continuity. *AdditiveResidual* and its variation, *PrescaledAdditiveResidual*, form convex combinations of the identity and transformed branches using a learnable scalar gate α: the former via interpolation $\alpha x + (1-\alpha)\mathrm{fn}(x)$ (with α constrained by a sigmoid), and the latter by premultiplying the input as $\frac{x+\mathrm{fn}(\alpha x)}{1+|\alpha|}$, where α unconstrained.

4. Unit testing of constrained layers

Despite theoretical guarantees, empirical verification remains crucial to detect (i) numerical instabilities (e.g., floating-point precision errors) and (ii) implementation discrepancies (e.g., padding mismatch, missing factor), ensuring orthogonality in practice. More details about our unit testing scheme are available in Appendix C.

Unit testing of convolutional layers. Orthogonality depends on training hyperparameters and convolution parameters (stride, group, dilation, transposition). We ensure correctness by combining two methods: (i) explicit SVD on Toeplitz matrices for precise validation on small inputs, and (ii) scalable spectral methods (from `conv.singular_values`) for practical, large-scale validation (as it uses parametrization-aware optimizations). All tests maintain singular value tolerance ranging from 10^{-4} to $5e^{-3}$ (for some methods).

We used both of these approaches in our unit tests. This enables us to ensure that the second method (which is faster and more scalable) is correct and effectively checks for layer orthogonality. We also added several unit tests to ensure that impossible theoretical configurations—as described in (Achour et al., 2022)—are rejected.

Unit testing of non-linear layers. To guarantee that activations and higher-order residual blocks do not silently violate the global 1-Lipschitz constraint, we complement the linear checks described above with a non-linear test suite based on the empirical Jacobian (computed with automatic differentiation) computed on randomly sampled and optimized tensors. This verified that the optimizer updates kept the block weights within the desired constraints.

Issues uncovered using unit-testing Crucially, these tests uncovered a flaw in the original HouseHolder activation (Singla & Feizi, 2022): the reflection missed a $1/\sqrt{2}$ normalization and was therefore $\sqrt{2}$-Lipschitz. Re-scaling the kernel collapses all singular values to 1 ± 10^{-5}, after which the layer satisfies our criteria.

Overall, this test bank was of precious use to confirm that all parameters can be combined in practice (i.e., A strided-grouped-transposed convolution with a dilation factor is still orthogonal). The library achieves 94% test coverage overall.

5. Conclusion

Orthogonium unifies a decade of advances in orthogonal and Lipschitz-constrained layers into a single, efficient, and comprehensive PyTorch library. By providing native support for strided, dilated, grouped, and transposed convolutions—alongside rigorous validation and optimized code—it significantly reduces implementation overhead, fosters reliable experimentation, and promotes adoption in critical applications such as certified robustness, generative modeling, and stable recurrent architectures. Furthermore, open-sourcing Orthogonium facilitates rigorous testing and validation by the broader community, uncovering subtle implementation errors and enabling ongoing verification and improvements. Orthogonium thus serves as a foundational resource, bridging theory and practice to enable scalable,

robust, and provably stable deep learning architectures.

Acknowledgements

This work was carried out within the DEEL project,[1] which is part of IRT Saint Exupéry and the ANITI AI cluster. The authors acknowledge the financial support from DEEL's Industrial and Academic Members and the France 2030 program – Grant agreements n°ANR-10-AIRT-01 and n°ANR-23-IACL-0002.

References

Achour, E. M., Malgouyres, F., and Mamalet, F. Existence, stability and scalability of orthogonal convolutional neural networks. Journal of Machine Learning Research, 23 (208):1–56, 2022.

Anil, C., Lucas, J., and Grosse, R. Sorting out lipschitz function approximation. In International Conference on Machine Learning, pp. 291–301. PMLR, 2019.

Araujo, A., Havens, A. J., Delattre, B., Allauzen, A., and Hu, B. A unified algebraic perspective on lipschitz neural networks. In The Eleventh International Conference on Learning Representations, 2023.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In International conference on machine learning, pp. 214–223. PMLR, 2017.

Bansal, N., Chen, X., and Wang, Z. Can we gain more from orthogonality regularizations in training deep networks? Advances in Neural Information Processing Systems, 31, 2018.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In International conference on machine learning, pp. 573–582. PMLR, 2019.

Björck, Å. and Bowie, C. An iterative algorithm for computing the best estimate of an orthogonal matrix. SIAM Journal on Numerical Analysis, 8(2):358–364, 1971.

Boissin, T., Mamalet, F., Fel, T., Picard, A. M., Massena, T., and Serrurier, M. An adaptive orthogonal convolution scheme for efficient and flexible cnn architectures. arXiv preprint arXiv:2501.07930, 2025.

Boroojeny, A. E., Telgarsky, M., and Sundaram, H. Spectrum extraction and clipping for implicitly linear layers. In International Conference on Artificial Intelligence and Statistics, pp. 2971–2979. PMLR, 2024.

[1] https://www.deel.ai/

Cayley, A. Sur quelques Propriétés des Déterminants Gauches, pp. 332–336. Cambridge Library Collection - Mathematics. Cambridge University Press, 1846.

Delattre, B., Barthélemy, Q., Araujo, A., and Allauzen, A. Efficient bound of lipschitz constant for convolutional layers by gram iteration. In International Conference on Machine Learning, pp. 7513–7532. PMLR, 2023.

Delattre, B., Barthélemy, Q., and Allauzen, A. Spectral norm of convolutional layers with circular and zero paddings. arXiv preprint arXiv:2402.00240, 2024.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255. Ieee, 2009.

Ding, X., Zhang, X., Han, J., and Ding, G. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11963–11975, 2022.

Fazlyab, M., Robey, A., Hassani, H., Morari, M., and Pappas, G. Efficient and accurate estimation of lipschitz constants for deep neural networks. Advances in neural information processing systems, 32, 2019.

Grishina, E., Gorbunov, M., and Rakhuba, M. Tight and efficient upper bound on spectral norm of convolutional layers. arXiv preprint arXiv:2409.11859, 2024.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. Advances in neural information processing systems, 30, 2017.

Hu, K., Leino, K., Wang, Z., and Fredrikson, M. A recipe for improved certifiable robustness. In The Twelfth International Conference on Learning Representations, 2023.

Kiani, B., Balestriero, R., LeCun, Y., and Lloyd, S. projunn: efficient method for training deep networks with unitary matrices. Advances in Neural Information Processing Systems, 35:14448–14463, 2022.

Kingma, D. P. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. Advances in neural information processing systems, 31, 2018.

LaPlace, P. S. Théorie analytique des probabilités. Courcier, 1820. URL http://eudml.org/doc/203444.

Li, Q., Haque, S., Anil, C., Lucas, J., Grosse, R. B., and Jacobsen, J.-H. Preventing gradient attenuation in lipschitz constrained convolutional networks. Advances in neural information processing systems, 32, 2019.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11976–11986, 2022.

Pauli, P., Wang, R., Manchester, I. R., and Allgöwer, F. Lipschitz-bounded 1d convolutional neural networks using the cayley transform and the controllability gramian. In 2023 62nd IEEE Conference on Decision and Control (CDC), pp. 5345–5350. IEEE, 2023.

Pauli, P., Gramlich, D., and Allgöwer, F. Lipschitz constant estimation for general neural network architectures using control tools. arXiv preprint arXiv:2405.01125, 2024.

Prach, B. and Lampert, C. H. Almost-orthogonal layers for efficient general-purpose lipschitz networks. In European Conference on Computer Vision, pp. 350–365. Springer, 2022.

Qi, H., You, C., Wang, X., Ma, Y., and Malik, J. Deep isometric learning for visual recognition. In International conference on machine learning, pp. 7824–7835. PMLR, 2020.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pp. 234–241. Springer, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4510–4520, 2018.

Sedghi, H., Gupta, V., and Long, P. M. The singular values of convolutional layers. In International Conference on Learning Representations, 2018.

Senderovich, A., Bulatova, E., Obukhov, A., and Rakhuba, M. Towards Practical Control of Singular Values of Convolutional Layers. Advances in Neural Information Processing Systems, 35:10918–10930, December 2022.

Serrurier, M., Mamalet, F., González-Sanz, A., Boissin, T., Loubes, J.-M., and Del Barrio, E. Achieving robustness in classification using optimal transport with hinge regularization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 505–514, 2021.

Singla, S. and Feizi, S. Fantastic four: Differentiable and efficient bounds on singular values of convolution layers. In International Conference on Learning Representations, 2021a.

Singla, S. and Feizi, S. Skew orthogonal convolutions. In International Conference on Machine Learning, pp. 9756–9766. PMLR, 2021b.

Singla, S. and Feizi, S. Improved techniques for deterministic l2 robustness. Advances in Neural Information Processing Systems, 35:16110–16124, 2022.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In International Conference on Learning Representations, 2014.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning, pp. 6105–6114. PMLR, 2019.

Trockman, A. and Kolter, J. Z. Orthogonalizing convolutional layers with the cayley transform. In International Conference on Learning Representations, 2021.

Virmaux, A. and Scaman, K. Lipschitz regularity of deep neural networks: analysis and efficient estimation. Advances in Neural Information Processing Systems, 31, 2018.

Wang, R. and Manchester, I. Direct parameterization of lipschitz-bounded deep networks. In International Conference on Machine Learning, pp. 36093–36110. PMLR, 2023.

Wang, Z., Hu, B., Havens, A. J., Araujo, A., Zheng, Y., Chen, Y., and Jha, S. On the scalability and memory efficiency of semidefinite programs for lipschitz constant estimation of neural networks. In The Twelfth International Conference on Learning Representations, 2024.

Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., and Pennington, J. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In International Conference on Machine Learning, pp. 5393–5402. PMLR, 2018.

A. Orthogonalization methods available for 2D weights matrices

A.1. QR Factorization via Modified Gram–Schmidt

We implement the Modified Gram–Schmidt (MGS) algorithm (LaPlace, 1820) for numerically stable QR factorization. Starting from $W = [w_i]_{i=0}^{C-1} \in \mathbb{R}^{C\times C}$, we orthogonalize one column at a time, correcting rounding errors at each step:

To enforce a unique factorization and improve stability, we post-multiply Q by $\mathrm{sign}(\mathrm{diag}(R))$ so that all diagonal entries of R are positive.

A.2. Cayley Transform

The Cayley transform (Cayley, 1846) maps any skew-symmetric A ($A^T = -A$) to an orthogonal Q via:

$$Q = (I - A)(I + A)^{-1}$$

For rectangular $W \in \mathbb{R}^{M\times C}$ ($M \geq C$), we follow the partitioning and "augmented" Cayley of (Pauli et al., 2023):

Algorithm 1 Augmented Cayley Transform

Require: $W \in \mathbb{R}^{M\times C}$
Ensure: $\hat{W} \in \mathbb{R}^{M\times C}$, orthogonal columns
procedure CAYLEY TRANSFORM(W)
  Partition $W = \begin{bmatrix} U \\ V \end{bmatrix}$, $U \in \mathbb{R}^{C\times C}$
  $A \leftarrow U - U^T + V^T V$ {not strictly skew but yields correct block}
  $B \leftarrow (I + A)^{-1}$
  $\hat{W}_1 \leftarrow B\,(I - A)$, $\hat{W}_2 \leftarrow -2\,V\,B$
  $\hat{W} \leftarrow [\,\hat{W}_1;\, \hat{W}_2\,]$
  return $\hat{W}$
end procedure=0

Matrix inversion can be a bottleneck, so in practice we cache and reuse factorizations where possible.

A.3. Exponential Map

Using the Lie-group exponential $\exp(A)$ of a skew-symmetric $A = W - W^T$ yields an orthogonal matrix since $\exp(A)^T = \exp(-A)$. We normalise A by its spectral norm to avoid overflow and truncate the power series after p terms (Singla & Feizi, 2021b):

Algorithm 2 Exponential Map with Spectral Normalization

Require: $W \in \mathbb{R}^{C\times C}$, $p \in \mathbb{N}$
Ensure: $\hat{W} = \exp(A)$ with $A = -A^T$
procedure LIPSCHITZ EXPONENTIAL(W, p)
  $A \leftarrow W - W^T$
  $\hat{A} \leftarrow A/\|A\|_2$ {spectral normalization}
  $\hat{W} \leftarrow I_C$, $\hat{A}_k \leftarrow I_C$
  for $k = 1, \ldots, p$ **do**
    $\hat{A}_k \leftarrow \frac{1}{k}\,\hat{A}_k\,\hat{A}$
    $\hat{W} \leftarrow \hat{W} + \hat{A}_k$
  end for
  return $\hat{W}$
end procedure=0

This method parametrizes only $\mathrm{SO}(C)$ and its accuracy depends on p.

A.4. Cholesky Decomposition

Following (Hu et al., 2023), we form the Gram matrix $C = WW^T + \varepsilon I$ ($\varepsilon > 0$ for PD), compute its Cholesky $C = LL^T$, and solve

$$L\,\hat{W} = W \quad \Longrightarrow \quad \hat{W} = L^{-1}W,$$

which enforces $\hat{W}\,\hat{W}^T = I$. This triangular solve is $O(C^3)$ but highly tuned in modern BLAS libraries.

A.5. Björck–Bowie Iterative Projection

The Björck–Bowie algorithm (Björck & Bowie, 1971) finds the nearest orthogonal matrix by fixed-point iteration

$$W_{t+1} = (1 + \beta)\,W_t - \beta\,W_t W_t^\top W_t, \quad \beta \in \left(0, \tfrac{1}{2}\right].$$

With spectral normalization of W_0 and $\beta = \frac{1}{2}$, convergence is fast: 12–25 iterations suffice in practice (Anil et al., 2019). We reuse a cached power-iteration estimate of the top singular vector across updates to further accelerate each step.

B. Technical details of improved orthogonal convolution methods

B.1. Improving BCOP (AOC)(Boissin et al., 2025)

Orthogonium introduces *Adaptive Orthogonal Convolution* (AOC),(Boissin et al., 2025), combining the strengths of BCOP (Li et al., 2019) and Reshaped Kernel Orthogonalization (RKO) (Serrurier et al., 2021). BCOP constructs explicit orthogonal convolution kernels by composing elementary orthogonal building blocks (such as 1×1, 1×2, and 2×1 convolutions), but originally lacks support for advanced convolutional operations like stride, dilation, transposition, or grouped convolutions. Conversely, RKO supports native

striding but typically achieves only approximate orthogonality.

AOC addresses these limitations by integrating BCOP and RKO into a single orthogonal convolutional kernel. Specifically, given a desired stride $s = k$, AOC defines the convolutional kernel as:

$$\text{AOC} = \text{RKO} \circledast \mathbf{K}_{\text{BCOP}} \tag{1}$$

The resulting kernel maintains strict orthogonality and explicitly supports stride, dilation, grouping, and transposed convolutions natively. Crucially, orthogonality is preserved through a careful choice of internal channel dimensions, ensuring both flexibility and computational efficiency (Boissin et al., 2025).

AOC is rigorously proven orthogonal for any valid configuration ($k \geq s$), (Achour et al., 2022), offering significant practical advantages over existing methods that rely on computationally expensive reshaping or Fourier-based operations.

Native Strided Convolution. Unlike prior methods that emulate striding via tensor reshaping—leading to substantial computational overhead—AOC implements native striding. This approach avoids the exponential computational complexity of reshaped methods, making orthogonal convolutions feasible for large-scale applications.

Native Transposed Convolution. By explicitly constructing orthogonal kernels, AOC naturally supports transposed convolutions, crucial for architectures requiring learnable upsampling such as U-Nets (Ronneberger et al., 2015) and Variational Autoencoders (VAEs) (Kingma, 2013).

Native Grouped Convolution. AOC efficiently supports grouped convolutions, widely used in contemporary models such as EfficientNet and ResNeXt, by independently orthogonalizing groups within the convolutional layer.

Dilation. Orthogonality under dilation follows naturally from AOC's explicit kernel construction, providing enlarged receptive fields without additional parameter overhead or loss of orthogonality.

These native implementations allow AOC to maintain a minimal overhead (approximately 10%) compared to unconstrained convolutional models, even at ImageNet scales.

In this section, we will explore how the content of this paper can be used to improve existing layers from the state of the art.

B.2. Improving skew orthogonal convolution (SOC)(Singla & Feizi, 2022)

This method, introduced by (Singla & Feizi, 2022) uses the fact that an exponential of a skew-symmetric matrix is orthogonal. The initial implementation builds a skew-symmetric kernel and computes the exponential convolution. However, without proper tools to compute the exponential of a convolution kernel, this exponential was computed implicitly for each input by using the Taylor expansion of the exponential (see Eq. (2)).

Theorem B.1 (Explicit conv exponential)**.** *We can use the block convolution operator* [2] *to compute explicitly the exponential of a kernel* $\mathbf{K}$*:*

$$x + \frac{\mathbf{K} * x}{1!} + \frac{\mathbf{K} * \mathbf{K} * x}{2!} + \dots \tag{2}$$

$$= \left(Id + \mathbf{K} + \frac{\mathbf{K} \circledast \mathbf{K}}{2!} + \frac{\mathbf{K} \circledast \mathbf{K} \circledast \mathbf{K}}{3!} + \dots \right) * x \tag{3}$$

Equation (3) shows that we can compute the exponential of a convolution kernel a single time, while the formulation in Eq. (2) needs to be done for each input x. In other words, we can apply one conv instead of n_{iter} convs. Note that the resulting kernel is then larger than the original one. In theory, this could unlock large speedups, but the gain is limited in practice as the implementation of convolution layers is optimized for small kernels and large images (Ding et al., 2022). However, the original implementation requires the storage of n_{iter} maps, whereas our implementation only one. This, in practice, unlocks larger networks and batch sizes.

Also, it is possible to handle a change in the number of channels and striding using a similar approach as AOC layers.

B.3. Improving SDP-based Lipschitz Layers (SLL) (Araujo et al., 2023)

SLL layer for convolutions, proposed in (Araujo et al., 2023), is a 1-Lipschitz layer defined as:

$$y = x - 2\mathbf{K}^T \star (\sigma(\mathbf{K} \star x + b))$$

Note that in the original paper, the equation is noted with product of two matrices $WT^{-\frac{1}{2}}$, for convolutions it represents toeplitz matrix, i.e. $WT^{-\frac{1}{2}} = \mathcal{K}$.

SLL layer does not natively support neither strides nor

[2] Block convolution operator allows to fuse the kernels of two convolutions to construct the kernel of a convolution equivalent to the composition of the two convolutions. It is defined in (Li et al., 2019), and an efficient implementation is available in (Boissin et al., 2025)

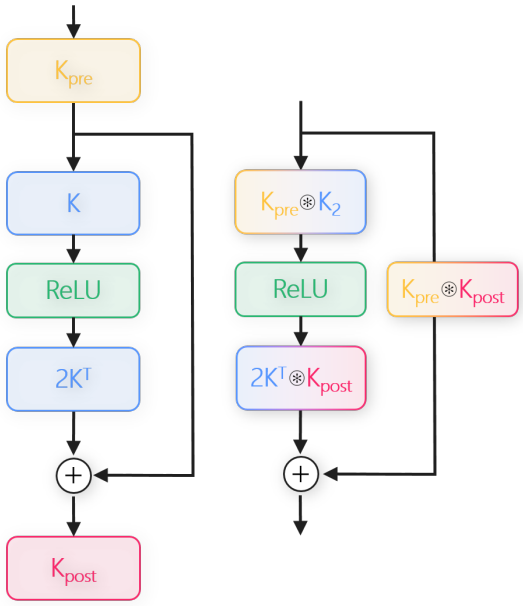

Figure 3: **The $\circledast$ can be used to enable $s \neq 1$ and $c_i \neq c_o$ configurations on SLL.** The flexibility of the $\circledast$ allows for operations resulting in a block with a similar structure as the original ResNet block.

changes in the channel size. We propose to use the $\circledast$ to derive a block, based on SLL, that supports stride and $c_i \neq c_o$, and can replace the strided convolutions of the residual branch in architectures like ResNet.

A natural first step is to append a strided convolution after a SLL block. This layer, $conv_{K_{post}} \circ SLL$, can then be fused in the SLL block thanks to block convolution operator [3]:

$$\begin{aligned} y =& \mathbf{K}_{post} \star_s (x - 2\mathbf{K}^T \star (\sigma(\mathbf{K} \star x + b))) \\ =& \mathbf{K}_{post} \star_s x - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma(\mathbf{K} \star x + b))) \end{aligned}$$

This allows to build a block based on SLL and that supports stride and channel changes. However, this creates an asymmetry between the convolution before the activation and the one after the activation (that has a larger kernel size).

We propose also to add a second convolution before the SLL block, $conv_{K_{post}} \circ SLL \circ conv_{K_{pre}}$ allowing better control over the kernel size of each convolution:

$$\begin{aligned} y =& \mathbf{K}_{post} \star_s \mathbf{K}_{pre} \star x \\ & - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma(\mathbf{K} \star \mathbf{K}_{pre} \star x + b))) \\ =& (\mathbf{K}_{post} \circledast \mathbf{K}_{pre}) \star_s x \\ & - 2(\mathbf{K}_{post} \circledast \mathbf{K}^T) \star_s (\sigma((\mathbf{K} \circledast \mathbf{K}_{pre}) \star x + b))) \end{aligned}$$

[3] as defined in (Li et al., 2019; Boissin et al., 2025)

The proposed block is still a 1-Lipschitz layer (as a composition of 1-Lipschitz and orthogonal layers), and support efficiently strides and changes of kernel sizes. A visual description is provided in Fig. 3. This approach is more efficient than the explicit construction that uses 3 distinct convolutions, as kernels are merged once per batch, and intermediate activations of extra convolutions do not need to be stored backward. Typically, when $\mathbf{K}$, $\mathbf{K}_{pre}$ and $\mathbf{K}_{post}$ are 2×2 convolutions, this results in a residual block with two 3×3 convolutions in one branch and a single 4×4 convolution (with stride 2) in the second. This is very similar to transition blocks found in typical residual networks.

B.4. Improving Sandwich Layers (Wang & Manchester, 2023)

Introduced by (Wang & Manchester, 2023), this approach aims to construct a 1-Lipschitz network globally rather than constraining each layer independently. In practice, this can be done either by (i) adding constraints between layers or (ii) creating layers that incorporate a non-linearity internally (a.k.a. sandwich layers). However, sandwich layers require an orthogonal matrix at their core. For convolutional layers, this is achieved by performing the orthogonalization of the layer in the Fourier domain, as described in the method from (Trockman & Kolter, 2021) and shown in their Algorithm 1.

Algorithm 3 Sandwich convolutional layer (from (Wang & Manchester, 2023))

Require: $h_{\text{in}} \in \mathbb{R}^{p \times s \times s}$, $P \in \mathbb{R}^{(p+q) \times q \times s \times s}$, $d \in \mathbb{R}^q$

$\hat{h}_{\text{in}} \leftarrow \text{FFT}(h_{\text{in}})$

$\Psi \leftarrow \text{diag}(e^d)$, $\begin{bmatrix} \tilde{A} & \tilde{B} \end{bmatrix}^* \leftarrow \text{Cayley}(\text{FFT}(P))$

$\hat{h}[:, i, j] \leftarrow \sqrt{2}\tilde{B}[:, :, i, j]\hat{h}_{\text{in}}[:, i, j]$

$\hat{h} \leftarrow \text{FFT}(\sigma(\text{FFT}^{-1}(\hat{h}) + b))$

$\hat{h}_{\text{out}}[:, i, j] \leftarrow \sqrt{2}\tilde{A}[:, :, i, j]\Psi\hat{h}[:, i, j]$

$h_{\text{out}} \leftarrow \text{FFT}^{-1}(\hat{h}_{\text{out}})$ =0

We can leverage AOC to construct the kernel of an orthogonal convolution, replacing the expensive operation performed in the Fourier domain. Thus, we can construct two kernels, $\mathbf{A}$ and $\mathbf{B}$, with appropriate constraints between the two and apply the rescaling and non-linearity directly in pixel space:

$$h_{\text{out}} = \sqrt{2}\mathbf{A}^\top \star \Psi\sigma\left(\sqrt{2}\Psi^{-1}\mathbf{B} \star h_{\text{in}} + b\right)$$

In practice this is done by constructing an orthogonal kernel with twice the number of channels that is split into two kernels, namely $\mathbf{A}$ and $\mathbf{B}$. This is expected to be more efficient since the use of the Fourier transform is costly for two reasons: first, it necessitates computation with complex

values; and second, the cost of the operation depends on the input size, which can be prohibitive in large-scale settings with 224×224 images. Consequently, our approach can make such a layer more scalable.

B.5. Extending Applicability to other methods.

Beyond the previously discussed approaches that show meaningful opportunities for improvement, our method can enhance a wide range of orthogonal convolutional layers. Specifically, we can incorporate our framework into any alternative orthogonal layers, enabling native support for strides in those layers. Furthermore, our approach can unlock features such as grouped convolutions, transposed convolutions, and dilations, broadening its utility and adaptability.

C. Technical details of our unit testing scheme

Evaluating the Lipschitz constant of a network Beyond the creation of a constrained layer, the evaluation of the Lipschitz constant of a layer is by itself an active field: early work used fast Fourier transform to evaluate a lower bound of the Lipschitz constant of a convolutional layer with circular padding (Sedghi et al., 2018). This work was later improved with a method that is quicker (Senderovich et al., 2022), supports other types of padding (Grishina et al., 2024), or allows the extraction of a larger part of the spectrum (Boroojeny et al., 2024). The work of (Delattre et al., 2023) (Delattre et al., 2024) allows us to compute a certifiable upper bound efficiently under different types of padding. It is worth recalling that inferring the global Lipschitz constant of a network given the Lipschitz constant of each layer is an NP-Hard problem(Virmaux & Scaman, 2018). Then, (Pauli et al., 2024; Fazlyab et al., 2019; Wang et al., 2024) aim to tackle using SDP (Semi-definite programming) tools. Our work can also contribute to this issue as the orthogonal layer allows a tighter product bound (ie. bound using the product of the Lipschitz constant of each layer to evaluate the constant of the whole network).

The need for an empirical evaluation of the Lipschitz constant of layers. Despite the theoretical guarantees ensuring orthogonality in our construction, empirical checks are necessary to confirm implementation correctness. Such verification prevents two types of issues:

1. **Checking of numerical Instabilities:** Issues arising from floating-point precision, such as those introduced by small epsilon values added to avoid division by zero.
2. **Checking for implementation discrepancies:** Differences between mathematical formalism and its translation to popular frameworks (e.g., SOC proofs assume circular padding, while its implementation uses zero padding, it is hard to determine how this difference affects the Lipschitz constant of such a layer).

Checking the orthogonality of a layer under stride, group, transposition, and dilation conditions. Orthogonality is sensitive to convolutional parameters such as stride, groups, dilation, and transposition, as well as training hyperparameters like learning rate, weight decay, and orthogonalization iterations. To robustly validate orthogonality, we combine two complementary approaches: (i) explicit singular-value decomposition (SVD) on convolution-induced Toeplitz matrices, ensuring exactness for small-scale inputs, and (ii) scalable spectral norm estimation via `conv.singular_values`, suitable for larger-scale practical validation. We thoroughly test diverse configurations—varying kernel sizes, strides, channel dimensions, and padding—ensuring all singular values remain within a stringent tolerance ranging from 10^{-4} to $5e^{-3}$ (for some methods).

The numerical stability and the convergence of an orthogonal layer is dependent on the training hyper-parameters: mainly the number of iterations used in most methods, but the learning rate and weight decay can also play a significant role. We then need an evaluation method that scales along with the convolution and that can be used at the end of each training. On the other hand, as scalable methods can be imperfect, we also need a method that computes very precise bounds without making any assumptions on the layer parameters (like padding, or stride). In order to overcome this, we tested our layers with two distinct methods:

Explicit SVD on Toeplitz Matrices: Using the impulse response approach, we construct the Toeplitz matrix for any padding and stride, allowing direct computation of singular values. This method, though accurate, is computationally expensive for large input images (in spite of full parallelization of the matrix's construction).

Evaluation from the `conv.singular_values` module: We use the more scalable methods (like (Delattre et al., 2023; 2024; Grishina et al., 2024)) from this module to check that the produced bounds from this module are valid.

We used both of these two approaches in our unit tests. This enables us to ensure that the second method (which is faster and more scalable) is correct to check that our layer is effectively orthogonal.

We tested multiple values for kernel size, stride, dilation, input channels, and output channels. For the kernel size, along with standard configurations of 3×3 and 5×5 kernels, we also covered cases for 1×1 kernels and even-sized kernels. For input/output channels, we covered various relevant inequalities (for instance, when $c_o > c_i s^2$ as indicated in (Achour et al., 2022)). We ran similar tests for transposed convolution (to the extent of what PyTorch allows: notably,

circular padding is not supported for transposed convolutions and could not be tested). Also, as the computation of the singular values using the explicit construction of the Toeplitz matrix is quite expensive, we used it on small 8×8 images; this is also a good way to check for padding issues, as the kernel size is not negligible with respect to the image size. All the checks over the singular values for both methods were done with a tolerance of $1e^{-4}$.

Unit testing of non-linear layers. To guarantee that activations and higher-order residual blocks do not silently violate the global 1-Lipschitz constraint, we complement the linear checks described above with a non-linear test-suite. For every candidate activation we sample small random tensors, build the full Jacobian with, and compute its spectral norm; checking $\max \sigma(J) \leq 1 + 10^{-4}$ and—when applicable— $\min \sigma(J) \geq 1 - 10^{-4}$, certifying both 1-Lipschitzness and orthogonality. A similar strategy is applied to parametrized layers such as SLL, CPL or AOL (Araujo et al., 2023; Xiao et al., 2018; Prach & Lampert, 2022) with the difference that before and after ten optimization steps we re-measure their Jacobian's spectral norm and assert it never exceeds 1, ensuring that optimizer updates cannot drift the block outside the desired constraint.

Because the Jacobian scales quadratically with the number of activations, inputs are restricted to 8×8 images or 64-d vectors—small enough for tractable SVD yet large enough to expose implementation bugs. Empirically, this design provokes failures in under three seconds on a laptop GPU and offers a pragmatic alternative to more expensive SDP-based constants.