A New Perspective for Graph Learning Architecture Design: Linearize Your Depth Away

Joël Mathys JMATHYS@ETHZ.CH and **Roger Wattenhofer** WATTENHOFER@ETHZ.CH
ETH Zurich

Editors: List of editors' names

Abstract

Designing effective graph learning architectures is central to making learning on structured and relational data feasible and scalable. Crucially, such designs must incorporate sufficient inductive bias to capture and leverage the graph topology. Simultaneously, they have to balance this objective with efficiently utilizing modern hardware, and remaining effectively trainable, even at scale. The current proposed architectures and paradigms range from message-passing neural networks on the graph topology to graph-informed transformers and virtual compute structures. Especially, the latter techniques often translate useful concepts and insights from graph theory in order to improve stability, mixing time, or bottlenecks. In this work, we highlight a linearization technique from the recently proposed Graph State-Space Model, as a powerful, general tool to design or improve graph learning architectures. At its core, the technique simplifies and reduces sequential computational depth and improves execution speed, while largely preserving trainability. Furthermore, the tool is versatile enough to be applied as a drop-in module across existing architectures. We showcase this flexibility by adapting Cayley Graph Propagation, yielding a simple, deeper and faster architecture.

Keywords: Graph Architectures, Graph Learning, State-Space Models, Linearization

1. Introduction

Designing effective graph learning architectures is central to making learning on structured and relational data both feasible and scalable. Graph-structured data appear in a wide range of domains, from molecular structures (Gilmer et al., 2017), social networks (Kipf and Welling, 2017), road networks (Neun et al., 2022) to combinatorial problems (Cappart et al., 2021). The core challenge lies in developing architectures that incorporate sufficient graph-aware inductive biases to leverage the structural information while also being able to capture sufficient domain-specific knowledge in a data-driven manner, all while remaining hardware-efficient and trainable at scale.

As a consequence, this has led to numerous proposed graph learning architectures, ranging from classical message-passing neural networks Gilmer et al. (2017) to graph transformers Dwivedi and Bresson (2021), walk-based approaches Chen et al. (2025) or virtual compute structures Wilson et al. (2024); Arnaiz-Rodriguez et al. (2022). In particular, the latter category often aims to translate insights from traditional graph theory, such as spectral propertiesMartinkus et al. (2022), expander constructions Deac et al. (2022), and hierarchical decompositions (Grötschla et al., 2024) to combat common issues in graph learning, such as oversmoothing and bottlenecks (Alon and Yahav, 2021). However, for

many of these theoretical guarantees to be translated, the architectures have to rely on sequential propagation steps to achieve the necessary information mixing, ultimately leading to increased sequential depth. This creates an inherent tension between the depth needed for sufficient expressivity and balancing training stability and computational efficiency.

In this work, we demonstrate that the linearization technique recently proposed by Graph State-Space Models (GSSM) Ceni et al. (2025) can serve as a flexible design tool for general graph architectures. At its core, linearization can transform sequential computation steps into efficient closed-form parallel computations, directly impacting and reducing the sequential depth while maintaining training stability. Rather than proposing a specialized architecture, we show how this technique can function as a drop-in enhancement for future and already existing graph learning architectures with minimal code modifications.

We validate our approach by proposing a new modification of Cayley Graph Propagation (CGP) (Wilson et al., 2024). Our adaptation demonstrates several benefits. Because there is no task-specific preprocessing, we can properly leverage precomputation for computational efficiency. Further, the linearization allows us to further facilitate the message exchange intended by the expander properties of the Cayley graph (Cayley, 1878). Finally, the closed-form parallel computation significantly reduces runtime overhead. We empirically validate our findings and demonstrate both the performance benefits and the computational efficiency gains of the linearized approach. Our work establishes the feasibility of applying linearization techniques to general graph architectures, providing practitioners with a valuable tool for improving computational efficiency without complete architectural overhauls. This positions linearization as a reusable design primitive that can advance the development of more scalable graph learning systems.

2. Background

Graph State-Space Models introduced by Ceni et al. (2025) draw inspiration from recent advances in state-space models for sequence modeling, which have demonstrated their effectiveness in capturing long-range dependencies while maintaining computational efficiency through parallel formulations. The GSSM framework extends this idea to graph-structured data by introducing modular linearized computation blocks of depth k with trainable parameters $\mathbf{W}$ and $\mathbf{B}$, adjacency matrix $\mathbf{A}$ and input $\mathbf{U}$.

$$\mathbf{X}_t = \mathbf{A}\mathbf{X}_{t-1}\mathbf{W} + \mathbf{U}_t\mathbf{B} \qquad\qquad \mathbf{Y} = \mathbf{MLP}(\mathbf{X}_k)$$

The key lies in the linearization technique that yields a fast closed-form computation which can leverage parallelism. This reduces sequential computational depth while minimally impacting trainability and stability within a block.

Cayley Graph Propagation (CGP) belongs to a family of architectures that take advantage of virtual compute structures designed to overcome limitations in message-passing neural networks (Deac et al., 2022). CGP leverages the mathematical properties of Cayley graphs, specifically their role as expander graphs of size n with guaranteed sparse connectivity, low $\mathcal{O}(\log n)$ diameter, and rapid mixing properties without structural bottlenecks. The architecture alternates between standard graph convolutions on the original topology and

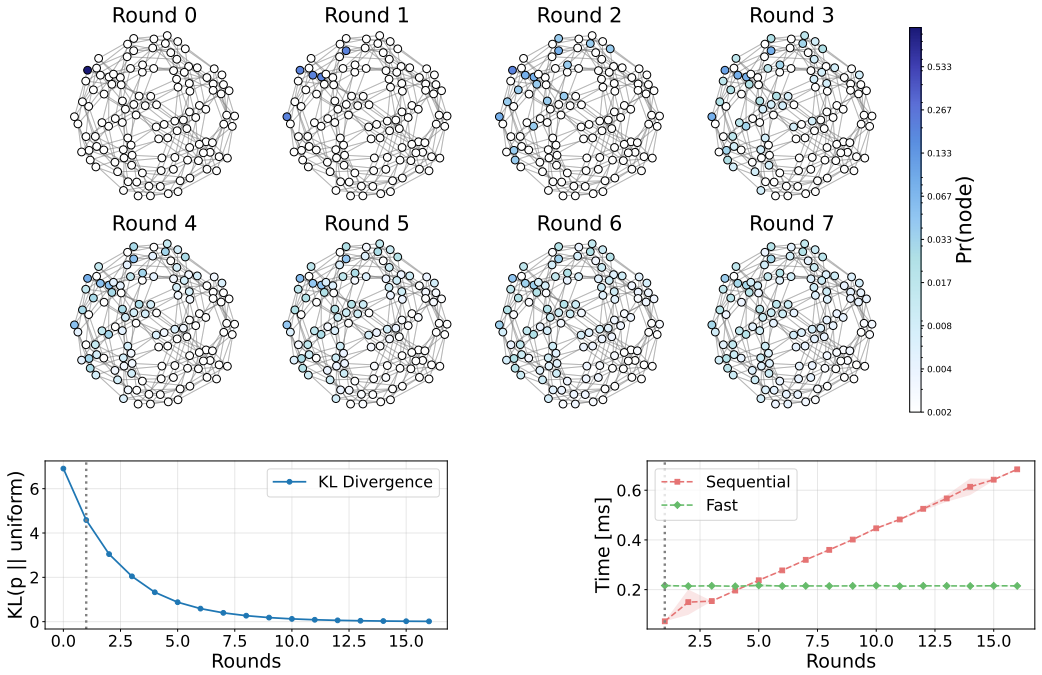

Figure 1: (Top) Visualization of probability mass diffusion on SL(5) from a single source node according to a random walk (with self-loops) of k steps. Due to the expander graph properties the mixing should converge withing $\mathcal{O}(\log n)$ steps. (Bottom Left) KL divergence to the uniform distribution decreases as information spreads, illustrating the expander graph's fast mixing. (Bottom Right) Runtime comparison between sequential computation (red) and closed-form computation (green). The grey dotted line indicates the effect of a single-step baseline used in CGP.

propagation on a virtual Cayley graph structure derived from the group $SL(k)$. Critically, this approach requires *no task-specific preprocessing*, making it broadly applicable.

3. Methodology

The linearization technique used in GSSM represents a more general design primitive well suited to improve existing graph architectures rather than limiting its use as a standalone model. The main advantage of the technique is to reduce the sequential computational depth through a closed-form parallel formulation. This is especially evident in classical message-passing architectures which require depth to expand their receptive fields, yet deep graph networks suffer from training instability, oversmoothing or oversquashing (Arnaiz-Rodriguez and Errica, 2025). Moreover, the technique is general enough to function as a drop-in replacement within existing architectural designs. We demonstrate this versatility by adapting Cayley Graph Propagation as our primary case study.

Our approach specifically modifies the expander propagation component within CGP while leaving all other architectural choices unchanged. In standard CGP, layers alternate between message passing on the original graph topology and propagation on the virtual

Cayley expander graph. We replace the *single-step* expander convolution with a linearized *multiple-step* formulation of depth T. A visualization of the architecture is shown in Figure 2 in the Appendix. Crucially, since Cayley expander graphs do not require task-specific preprocessing, we can precompute the required decomposition offline, making the closed-form linearization even more efficient.

This modification alters the information exchange dynamics within the architecture. The original CGP alternates between topologies at each layer, equally balancing the two by executing a single convolution. Our reformulation uses multiple convolutions on the expander graph instead. Since Cayley expander graphs require $\mathcal{O}(\log n)$ steps to uniformly mix information between nodes, the original standard formulation does not fully exploit these properties. By increasing the number of convolutions beyond a single step, we aim to enable more thorough information exchange. At the same time, the closed-form formulation adds minimal overhead to the computation, eliminating a potential sequential bottleneck. To further illustrate this process, we visualize the information diffusion process on a Cayley graph in Figure 1. The probability mass spreads following a random walk (with self-loops) of length k from an initial node across the expander structure over multiple steps. To further validate this approach empirically, we ablate our modified architecture on the MUTAG dataset (Morris et al., 2020), where the original CGP improved the most over the baseline GCN method. Table 1 shows that the linearized multi-step expander propagation consistently improves the original CGP formulation across different unrolling depths. For more details on the experimental evaluation we refer to the Appendix.

Table 1: Empirical results on the MUTAG dataset. Both CGP and CGP-SSM use the architecture by Wilson et al. (2024) with the GCN convolution as a base, CGP-SSM replaces the expander propagation with the GSSM linearization. We report the 95% confidence intervals computed over 50 seeds.

Model	CGP	Depth	MUTAG ↑
GCN			73.60 ± 3.15
CGP	✓		77.60 ± 3.17
CGP-SSM	✓	3	79.90 ± 2.61
CGP-SSM	✓	6	79.00 ± 2.62
CGP-SSM	✓	9	79.80 ± 2.58
CGP-SSM	✓	12	**80.90** ± 2.43
CGP-SSM	✓	15	79.70 ± 2.53
CGP-SSM	✓	100	79.90 ± 2.65

4. Conclusion

We demonstrate that the linearization technique of Graph State-Space Models serves as a general design tool that extends well beyond the standalone architectures. The technique's main benefit is to effectively reduce sequential computational depth through parallel formulations, without the training instabilities typically associated with deep non-linear sequential operations. Our adaptation of Cayley Graph Propagation validates this approach in practice, showing that even current architectures can be improved with minimal overhead. In addition, we showcase how linearization can be a way to enable a more thorough mixing of information on expander graphs while maintaining computational efficiency. This demonstrates that the linearization tool is an interesting perspective for general graph learning architecture designs towards developing more capable, efficient, and scalable graph learning.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2021. URL https://arxiv.org/abs/2006.05205.

Adrian Arnaiz-Rodriguez and Federico Errica. Oversmoothing, oversquashing, heterophily, long-range, and more: Demystifying common beliefs in graph machine learning, 2025. URL https://arxiv.org/abs/2505.15547.

Adrian Arnaiz-Rodriguez, Ahmed Begga, Francisco Escolano, and Nuria Oliver. Diffwire: Inductive graph rewiring via the lovász bound, 2022. URL https://arxiv.org/abs/2206.07369.

Ali Behrouz and Farnoosh Hashemi. Graph mamba: Towards learning on graphs with state space models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 119–130, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3672044. URL https://doi.org/10.1145/3637528.3672044.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/595. URL https://doi.org/10.24963/ijcai.2021/595. Survey Track.

Professor Cayley. Desiderata and suggestions: No. 2. the theory of groups: Graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878. ISSN 00029327, 10806377. URL http://www.jstor.org/stable/2369306.

Andrea Ceni, Alessio Gravina, Claudio Gallicchio, Davide Bacciu, Carola-Bibiane Schonlieb, and Moshe Eliasof. Message-passing state-space models: Improving graph learning with modern sequence modeling, 2025. URL https://arxiv.org/abs/2505.18728.

Dexiong Chen, Till Hendrik Schulz, and Karsten Borgwardt. Learning long range dependencies on graphs via random walks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kJ5H7oGT2M.

Corinna Coupette, Jeremy Wayland, Emily Simons, and Bastian Rieck. No metric to rule them all: Toward principled evaluations of graph-learning datasets. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=XbmBNwrfG5.

Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=IKevTLt3rT.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021. URL https://arxiv.org/abs/2012.09699.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Florian Grötschla, Joël Mathys, Robert Veres, and Roger Wattenhofer. Core-GD: A hierarchical framework for scalable graph visualization with GNNs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vtyasLn4RM.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Nicolas Keriven. Not too little, not too much: a theoretical analysis of graph (over)smoothing. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1l2bp4YwS.

Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators, 2022. URL https://arxiv.org/abs/2204.01613.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Moritz Neun, Christian Eichenberger, Henry Martin, Markus Spanring, Rahul Siripurapu, Daniel Springer, Leyan Deng, Chenwang Wu, Defu Lian, Min Zhou, Martin Lumiste, Andrei Ilie, Xinhua Wu, Cheng Lyu, Qing-Long Lu, Vishal Mahajan, Yichao Lu, Jiezhang Li, Junjun Li, Yue-Jiao Gong, Florian Grötschla, Joël Mathys, Ye Wei, He Haitao, Hui

Fang, Kevin Malm, Fei Tang, Michael Kopp, David Kreil, and Sepp Hochreiter. Traffic4cast at neurips 2022 – predict dynamics along graph edges from sparse node data: Whole city traffic and eta from stationary vehicle detectors. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 251–278. PMLR, 28 Nov–09 Dec 2022. URL https://proceedings.mlr.press/v220/neun23a.html.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Matus Telgarsky. benefits of depth in neural networks. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR. URL https://proceedings.mlr.press/v49/telgarsky16.html.

JJ Wilson, Maya Bechler-Speicher, and Petar Veličković. Cayley graph propagation. In *The Third Learning on Graphs Conference*, 2024. URL https://openreview.net/forum?id=VaTfEDs6lE.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/wu19e.html.

Appendix A. Extended Overview

A.1. Related Work

State-Space Models Recent breakthroughs in sequence modeling, from S4 (Gu et al., 2022) to Mamba (Gu and Dao, 2024), demonstrate effectiveness in capturing long-range dependencies through linear parallel formulations. Behrouz and Hashemi (2024); Chen et al. (2025) adapt this approach by converting graphs to sequences via walk-based methods. In contrast, Ceni et al. (2025) formulate an SSM variant closer to graph convolutions with a focus on static and temporal graph data. In our work, we highlight the general applicability of the underlying linearization technique as a design tool.

Linearization Linearization aims to remove non-linearities in the computation flow. On the one hand, this can lead to more efficient closed-form formulations, and on the other it simplifies the underlying mechanisms enough to approach them with rigorous mathematical analysis (Keriven, 2022). The work of Wu et al. (2019) removed non-linearities, creating competitive linear models with computational savings. However, this was mainly considered as a standalone architecture rather than a design component, potentially limiting opportunities for stackable components that could increase performance and make it more generally applicable.

Depth The relationship between depth and width plays a special role in the graph community. Loukas (2020) establishes theoretical lower bounds showing that depth is crucial, especially when the width is small compared to overall graph size. Further, in classical graph convolutions depth is often the main driver for receptive field expansion (Liu et al., 2020). However, this is not the only role of depth in the computation (Telgarsky, 2016), it also needs to properly process and transform information. The fundamental challenge is that non-linearities at depth impose significant costs while depth remains theoretically necessary. The GSSM linearization technique offers one possible way to incorporate this as a design tool while balancing it with a convolutional graph bias.

A.2. Future Outlook

In the case of CGP the linearization technique emphasizes improved information mixing without the drawback of additional sequential compute depth, all while leveraging hardware-efficient computation. However, the computational speedup comes at the cost of increased memory requirements for using decomposed matrices ultimately needed for faster computation. Looking ahead, this linearization paradigm might open up several new research directions. Recent breakthroughs in advanced state-space model techniques ranging from S5 (Smith et al., 2023) to Mamba (Gu and Dao, 2024), relied on clever initialization, hardware-aware implementations and selective mechanisms, yet to fully translate to the graph domain. Finally, for graph practitioners, this raises intriguing possibilities about the future role of virtual graph structures: effectively balancing computational depth, training stability, and theoretical guarantees under computational efficiency.

A.3. Architecture Overview

Appendix B. Experimental Evaluation

Architecture and Setup: We used the original CGP implementation (Wilson et al., 2024) as our baseline, preserving all hyperparameters choices such as: early stopping with a patience of 100 epochs, trained for maximum 300 epochs, 4 layers, hidden dimension of 64, 50% dropout, Batchnorm. Our only modification for CGP-SSM replaced the single expander convolution with the linearized multi-step formulation. In addition, we initialized the W weight matrices with a spectral radius close to 1.0 following standard practices for numerical stability. This provided consistent improvements to CGP-SSM performance, although it was not decisive to improve wrt. to the CGP baseline. All MUTAG results are averaged over 50 independent runs and we report confidence intervals at 95%. Note that our reported results are lower than the numbers in the original CGP paper due to correcting a model selection procedure in the training loop that might implicitly leak information about the test set. Further, we want to note that many of the TU datasets are likely no longer ideal for evaluating modern learning approaches, as outlined in (Coupette et al., 2025). Our primary reason for ablating on MUTAG, besides it being the dataset where CGP improved the most wrt. to its base, is to remain as close as possible to the original CGP to illustrate the benefits of the linearization technique.

Timing Experiments: Runtime measurements were conducted on an A600 GPU. Each measurement consisted of 100 sequential graph inputs, repeated 10 times to com-

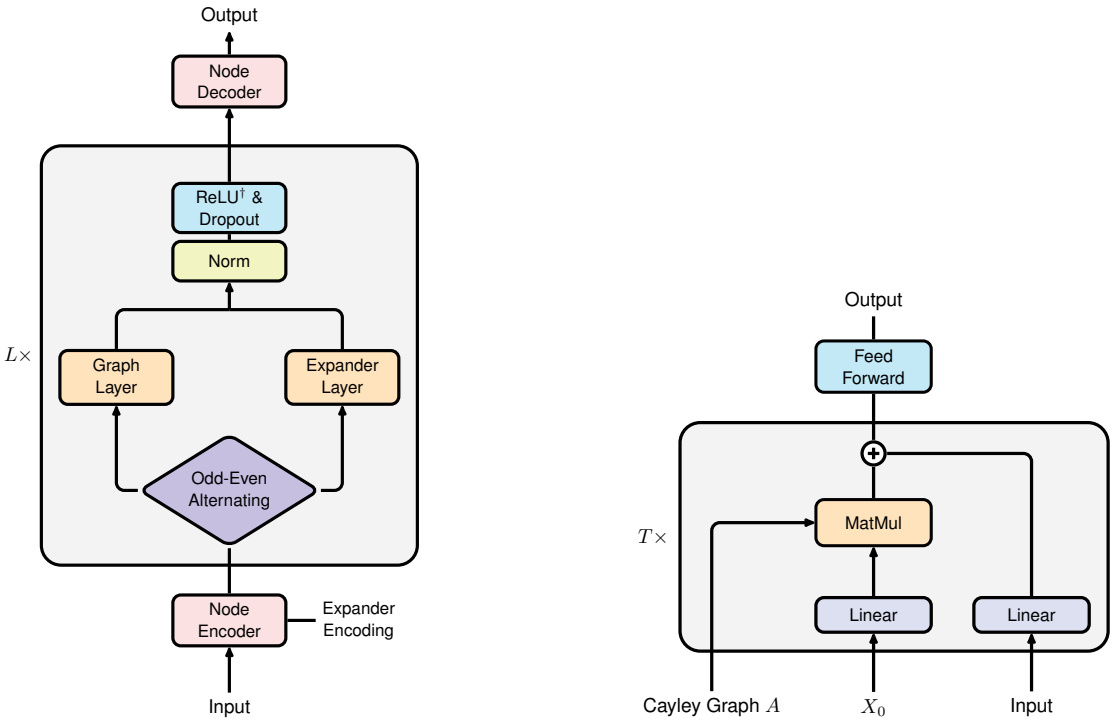

Figure 2: (Left) CGP alternates blocks of a single graph convolution on the original and expander topology. (Right) Our CGP-SSM replaces the single expander layer with T linearized convolutions. This reduces sequential depth while mixing more information.

pute standard deviations. We precomputed all components that were independent of the unrolling steps and input features for both variants.

Appendix C. Cayley Graphs

We include general statistics about the Cayley graphs used in our experiments in Table 2 and further visualizations for sizes 3,4,6 and 14.

Appendix D. Implementation

Following the notation and derivation from Ceni et al. (2025) we can write the GSSM in a blocks as follows with A as the degree normalized adjacency matrix $A = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$

Table 2: Overview over Cayley graphs generated by the group SL. The statistics were computed with the help of the networkx library.

SL(k)	Nodes	Edges	Avg. Degree	Diameter	Avg. Shortest Path Length	$\log_2 N$
SL(2)	6	24	2.00	3	1.80	2.58
SL(3)	24	96	4.00	4	2.35	4.58
SL(4)	48	192	4.00	6	3.11	5.58
SL(5)	120	480	4.00	6	3.75	6.91
SL(6)	144	576	4.00	6	3.99	7.17
SL(7)	336	1344	4.00	7	4.85	8.39
SL(8)	384	1536	4.00	8	5.02	8.58
SL(9)	648	2592	4.00	8	5.70	9.34
SL(10)	720	2880	4.00	10	5.97	9.49
SL(11)	1320	5280	4.00	10	6.60	10.37
SL(12)	1152	4608	4.00	10	6.47	10.17
SL(13)	2184	8736	4.00	10	7.37	11.09
SL(14)	2016	8064	4.00	11	7.27	10.98
SL(15)	2880	11520	4.00	12	8.04	11.49
SL(16)	3072	12288	4.00	12	8.18	11.58
SL(17)	4896	19584	4.00	12	8.41	12.26
SL(18)	3888	15552	4.00	12	8.24	11.92
SL(19)	6840	27360	4.00	13	9.10	12.74
SL(20)	5760	23040	4.00	14	9.04	12.49

with self-loops.

$$\begin{aligned} X_0 &= 0 \\ X_t &= AX_{t-1}W + UB \\ X_t &= \sum_{i=0}^{t} A^i U B W^i \end{aligned}$$

Further, if we assume that we can decompose $A = P\Lambda P^{-1}$ and $V = V\Sigma V^{-1}$ we can write out the closed form as follows where $*$ denotes the element-wise multiplication.

$$\begin{aligned} X_t &= P\left(\sum_{i=0}^{t-1} \Lambda^i P^{-1} U B V \Sigma^i\right) V^{-1} \\ X_t &= P\left(\sum_{i=0}^{t-1} \Lambda^i S \Sigma^i\right) V^{-1} \\ X_t &= P(S * \Lambda_{pow}^T \Sigma_{pow}) V^{-1} \end{aligned}$$

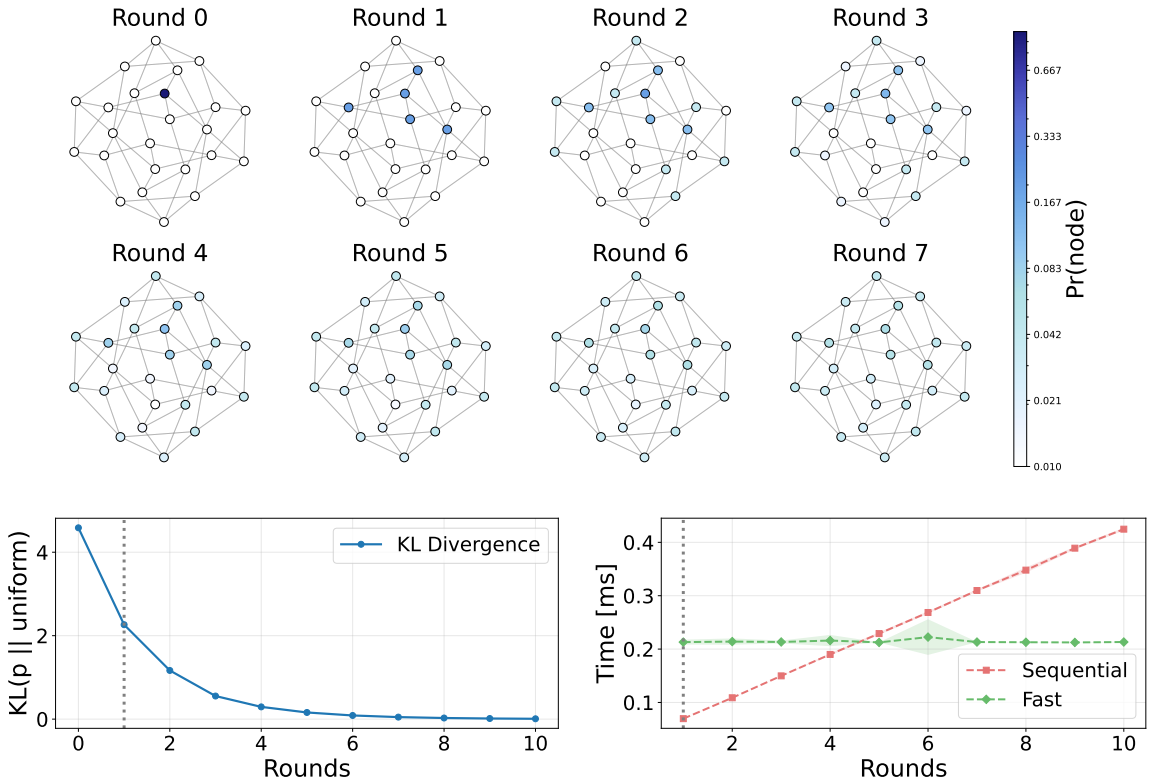

Figure 3: Cayley graph generated by SL(3). The top shows the first steps of the information diffusion and the bottom left shows the corresponding distance to the uniform distribution. On the right, the measured time to compute the expander convolution either sequentially or in closed form.

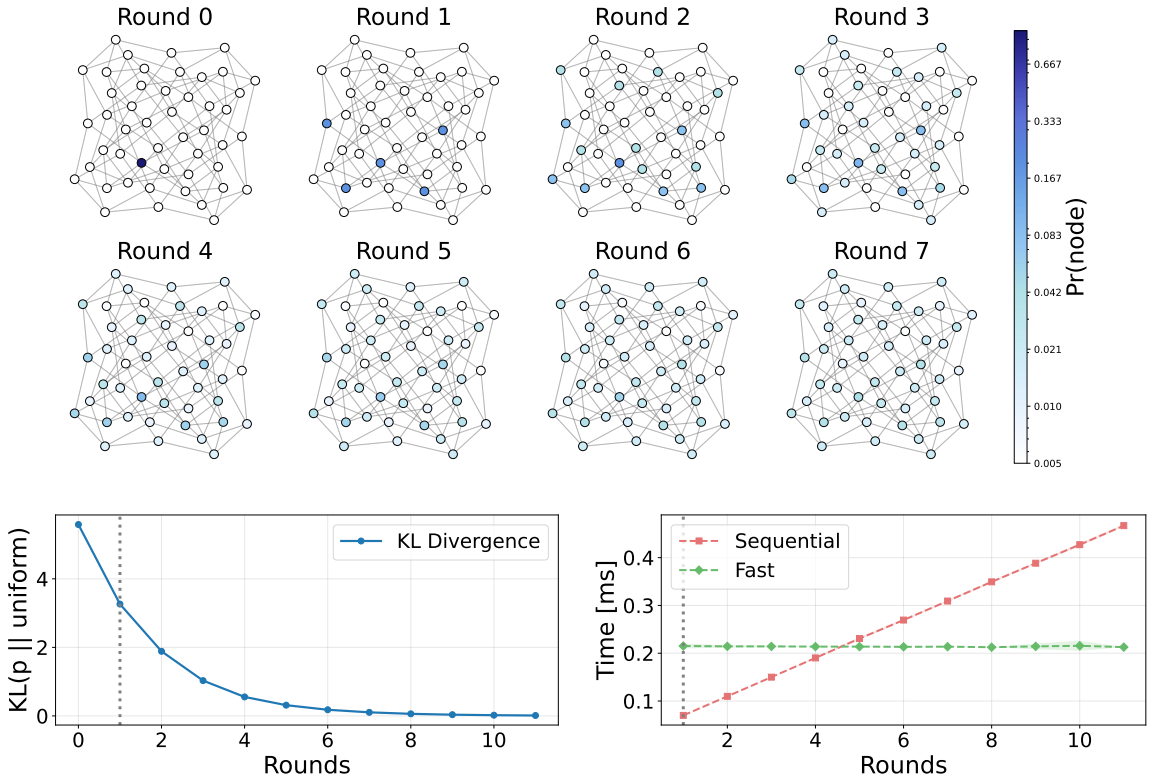

Figure 4: Cayley graph generated by SL(4). The top shows the first steps of the information diffusion and the bottom left shows the corresponding distance to the uniform distribution. On the right, the measured time to compute the expander convolution either sequentially or in closed form.

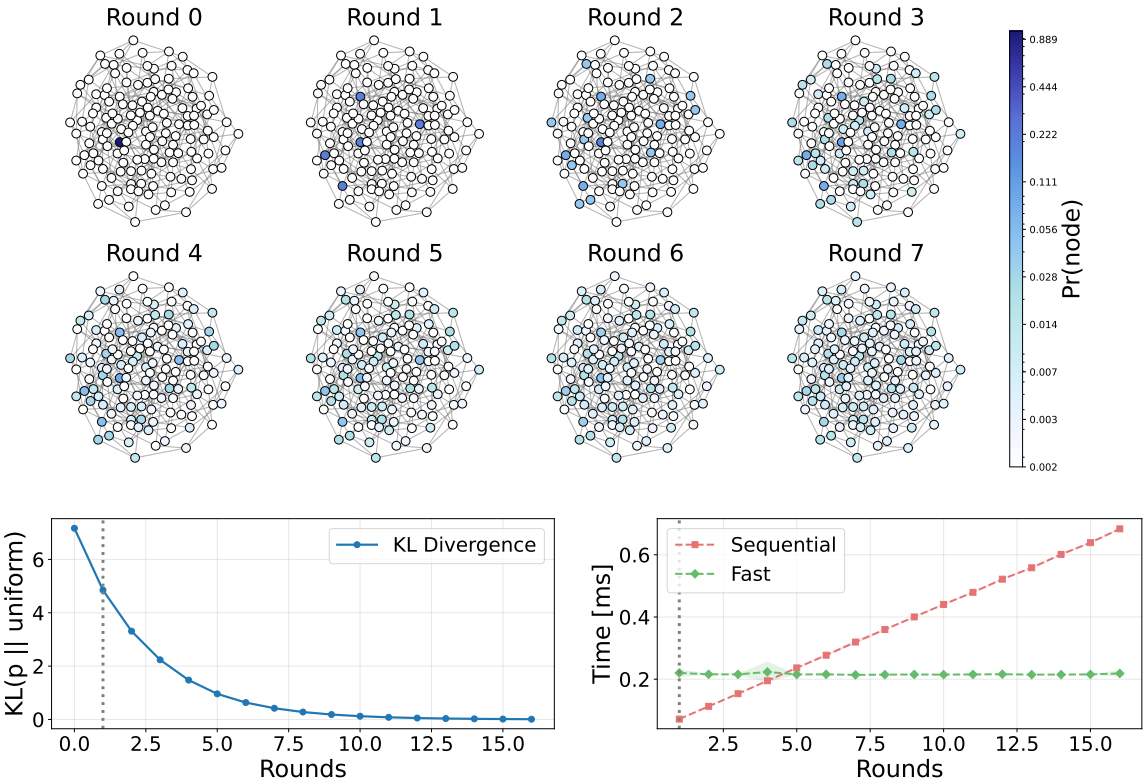

Figure 5: Cayley graph generated by SL(6). The top shows the first steps of the information diffusion and the bottom left shows the corresponding distance to the uniform distribution. On the right, the measured time to compute the expander convolution either sequentially or in closed form.

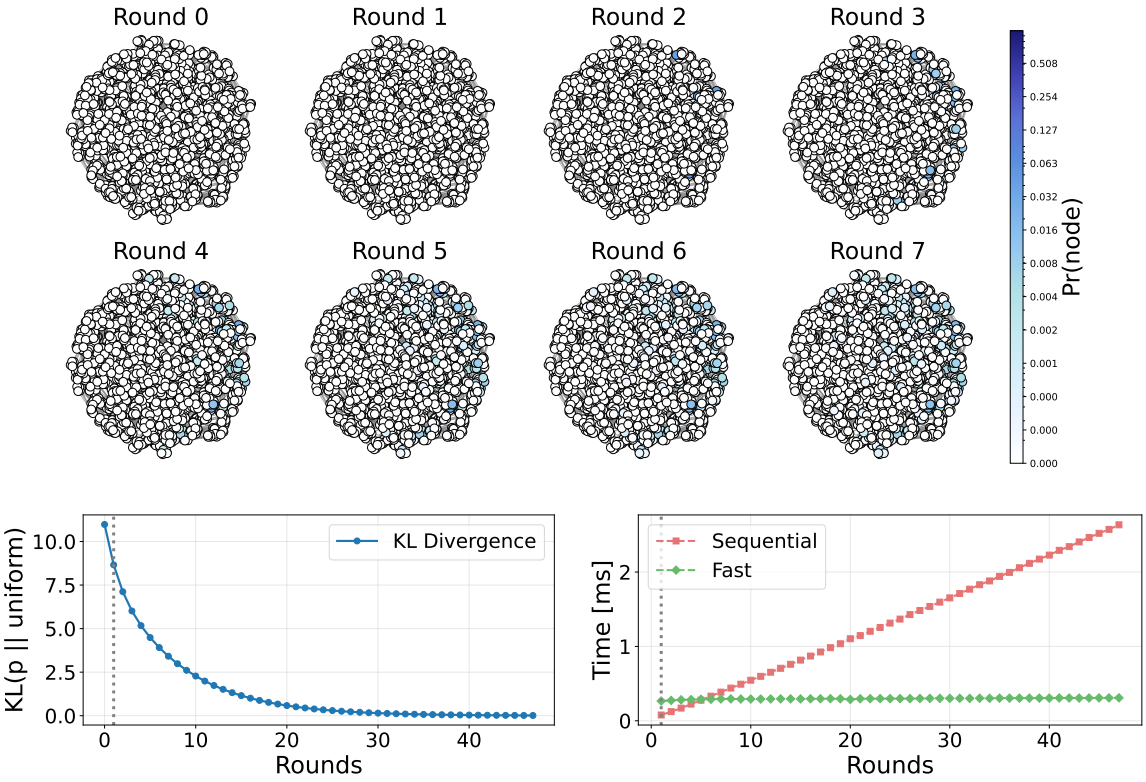

Figure 6: Cayley graph generated by SL(14). The top shows the first steps of the information diffusion and the bottom left shows the corresponding distance to the uniform distribution. On the right, the measured time to compute the expander convolution either sequentially or in closed form.

```python
import torch
from torch_geometric.nn.conv.gcn_conv import gcn_norm
from torch_geometric.nn import GCNConv

def edge_index_to_gso(edge_index):
    # GCN normalization + self-loops, then densify
    edge_index_n, edge_weight_n = gcn_norm(
        edge_index, add_self_loops=True, num_nodes=None
    )
    num_nodes = int(edge_index.max().item()) + 1
    A = torch.sparse_coo_tensor(edge_index_n, edge_weight_n,
                                (num_nodes, num_nodes)).to_dense()
    return A

class GraphSSMExample(torch.nn.Module):
    """
    x_{t+1} = A x_t W + U B
    Uses an explicit (dense) GSO A with GCN-style normalization + self-loops.
    """
    def __init__(self, hidden_dim):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.W = torch.nn.Parameter(torch.randn(hidden_dim, hidden_dim))
        self.B = torch.nn.Parameter(torch.randn(hidden_dim, hidden_dim))

    def dense_implementation(self, batch, num_steps):
        U = batch.x                                  # (n, d)
        A = edge_index_to_gso(batch.edge_index)      # (n, n)

        x = torch.zeros_like(U)                      # (n, d)
        for _ in range(num_steps):
            x = A @ x @ self.W + U @ self.B
        return x
    def gcn_implementation(self, batch, num_steps):
        U = batch.x
        device, dtype = U.device, U.dtype

        # GCN with self-loops + normalization;
        gcn = GCNConv(
            self.hidden_dim, self.hidden_dim,
            bias=False, add_self_loops=True, normalize=True).to(device)

        # Make GCN's internal linear be "x @ W" (note the transpose).
        gcn.lin.weight = torch.nn.Parameter(self.W.T.clone().to(device, dtype))

        x = torch.zeros_like(U)
        for _ in range(num_steps):
            x = gcn(x, batch.edge_index) + U @ self.B
        return x

    def fast_implmentation(self, batch, num_steps):
        A = edge_index_to_gso(batch.edge_index)              # (n, n)
        sig, P = torch.linalg.eigh(A)                        # A = P diag(sig) P^T

        sig_w, V = torch.linalg.eig(self.W)                  # W = V diag(sig_w) V^{-1}
        V_inv = torch.linalg.inv(V)

        U = batch.x
        # Transform U B into both eigen-bases
        BV = (self.B.to(torch.cfloat) @ V.to(torch.cfloat))
        S  = (P.T @ U).to(torch.cfloat) @ BV                 # (n, d)

        T = int(num_steps)
        k = torch.arange(T, device=U.device)

        # Powers across time for each eigenvalue
        A_pow = sig.unsqueeze(0).pow(k.unsqueeze(1))         # (T, n)
        W_pow = sig_w.unsqueeze(0).pow(k.unsqueeze(1))       # (T, d)

        weights = A_pow.to(torch.cfloat).transpose(0, 1) @ W_pow  # (n, d)

        X_hat = S * weights
        X = P.to(torch.cfloat) @ X_hat @ V_inv.to(torch.cfloat)
        return X.real.to(U.dtype)
```

Listing 1: Example implementations of the Graph State-Space Model using torch and pytorch geometric. This assumes that U remains constant over time, for the original, both formulation and fast parallel implementation of the generalized case, refer to the works of Ceni et al. (2025).