
Learning Lattice Variables for Matrix-Free Wilson-Dirac Preconditioning in Lattice QCD

Adam Shaw *

Argonne National Laboratory

Srinivas Eswar

Argonne National Laboratory

William Detmold

Massachusetts Institute of Technology

Phiala Shanahan

Massachusetts Institute of Technology

Xiaoye Sherry Li

Lawrence Berkeley National Laboratory

Yang Liu

Lawrence Berkeley National Laboratory

Yixuan Sun[†]

Argonne National Laboratory

Abstract

Solving large, sparse, and ill-conditioned linear systems dominates the computational cost of lattice quantum chromodynamics (LQCD). Preconditioners effectively accelerate the convergence of iterative solvers but generally need careful design and tuning. This work develops an encoder–representation–decoder framework to produce a matrix-free neural preconditioner for the Wilson-Dirac operator. The encoder maps the gauge configuration U to a representation V , which the decoder turns into a Hermitian positive-definite (HPD) preconditioner M^{-1} for the Krylov solver. Because the representation consists of matrix-valued lattice fields, the decoder is assembled from lattice operators and never forms the actual preconditioning matrix—a computationally prohibitive operation for typical systems. No learned quantity in either the encoder or the decoder is tied to the lattice volume, so a trained encoder–decoder pair applies unchanged at any lattice size. Across five encoder–decoder pairs, we train the framework using gauge configurations of lattice size $L^4 = 8^4$ against the unrolled preconditioned Conjugate Gradient (PCG) objective and show that the generated preconditioners effectively reduce the number of iterations required for convergence, achieving a $7.8\times$ reduction compared to the unpreconditioned case. In addition, we directly apply the trained encoder–decoder pairs to a larger system of $L^4 = 16^4$, where the convergence behavior and relative ordering are preserved ($\sim 8\times$ iteration reduction) without retraining.

1 Introduction

LQCD requires the repeated solution of large, sparse linear systems

$$A\psi = b, \quad A = (D^\dagger D)[U], \quad b = D^\dagger[U]\eta, \quad (1)$$

where $D[U]$ is the discretized Wilson-Dirac operator on a background gauge configuration $U = \{U_\mu(x)\}_{x \in \Lambda, \mu=1,\dots,4}$ with $U_\mu(x) \in \text{SU}(3) \subset \mathbb{C}^{3 \times 3}$, ψ is the quark field, η is a fermion source, Λ

*Also at University of Southern California

[†]Corresponding author.

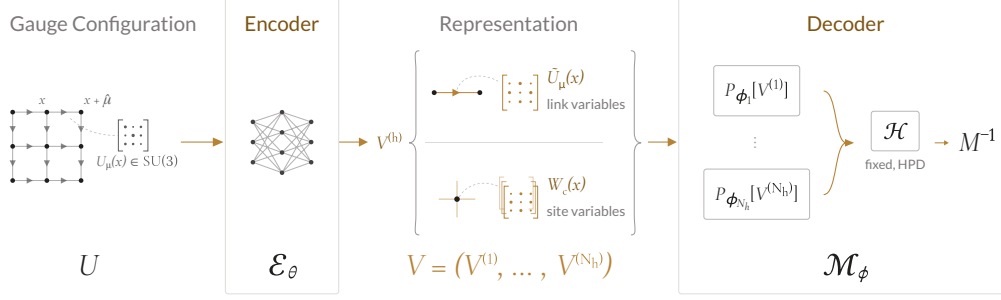


Figure 1: The encoder–representation–decoder pipeline. An encoder maps the data defining the system to a representation of site and/or link variables, which the decoder uses to build a preconditioner.

denotes the finite four-dimensional hypercubic lattice of sites, and $\mu \in \{1, \dots, 4\}$ labels a lattice direction ($\hat{\mu}$ denotes the unit lattice vector in direction μ) [Wilson, 1974]. These systems are extremely high dimensional ($\dim(\Lambda) \gtrsim 10^8$) and ill-conditioned in production-scale calculations, hence solving them dominates the computational cost of computing observables [Joó et al., 2019].

Preconditioners, often denoted as M^{-1} , accelerate the convergence of solving the Dirac equation by reducing the condition number of the linear system. However, classical preconditioners require careful design and tuning, often incurring a substantial setup cost for each new system [Saad, 2003]. Meanwhile, learned preconditioners amortize this into a one-time training cost, producing M^{-1} for a new system in a single forward pass [Häusner et al., 2024, Pfahler et al., 2026, Sun et al., 2026, Cali et al., 2023, Lehner and Wettig, 2023]. Due to the high dimensionality, learned preconditioners for the Dirac equation in LQCD must be matrix-free—circumventing storing the entire matrix at once—and ideally applicable across variations of the lattice dimensionality. When paired with PCG solvers, the preconditioners additionally need to be fixed linear HPD operators.

Inspired by the matrix-free neural preconditioner for the U(1) Schwinger model [Sun et al., 2026], we propose a general encoder–representation–decoder framework that constructs preconditioners for the Dirac operator associated with each SU(3) gauge configuration, shown in Figure 1. Having the unrolled PCG residuals [Nytke et al., 2025] as the training objective, this framework is effective across four encoder and four decoder architectures, significantly reducing the number of iterations needed for convergence compared to the unpreconditioned case. Trained on gauge configuration data with lattice size $L^4 = 8^4$, the framework provides up to a $7.8\times$ reduction in the number of iterations, and is immediately applicable to a larger lattice size ($L^4 = 16^4$) with up to an $8.1\times$ reduction, preserving the convergence acceleration without further training or tuning.

2 Related work

Preconditioning. Preconditioning replaces a linear system with an equivalent one that an iterative method solves faster. When using the CG solver, the iteration count grows as $O(\sqrt{\kappa(A)})$ where $\kappa(\cdot)$ is the condition number; hence, an effective preconditioner M^{-1} will be such that $M^{-1}A$ is better conditioned than A alone. Classical preconditioners such as incomplete Cholesky and even-odd (Schur complement) decomposition are built directly from the entries of A and must be rebuilt for every new system (here every gauge configuration) [Saad, 2003], a significant setup cost that affects the speed of these solves. Adaptive multigrid (AMG) [Trottenberg et al., 2000, Brandt and Livne, 2011], the current state-of-the-art for the Wilson-Dirac equation [Frommer et al., 2014, Clark et al., 2010, 2016, Brezina et al., 2006], has such a setup cost (grid hierarchy, null-vector computations, and associated hyperparameters) and must be re-tuned for each new lattice size for optimal results.

Learned preconditioners. Learned preconditioners replace this per-system construction with a one-time training cost, producing M^{-1} for a new system in a single forward pass [Häusner et al., 2024]. However, existing learned preconditioners fall short in different ways. Some form A or M^{-1} explicitly [Cali et al., 2023, Suppl et al., 2019], and others must be retrained for each lattice size [Häusner et al., 2024], both impractical at production scale. Recent work [Sun et al., 2026] learns

a map $U_\mu(x) \mapsto \tilde{U}_\mu(x)$ (a new link variable) and defines $M^{-1} = (D^\dagger D)[\tilde{U}]$, which is matrix-free and transfers across volumes. However, the authors work only with $U(1)$ gauge links and their preconditioner (decoder) is restricted to the same Dirac normal operator $D^\dagger D$ the system is defined by.

3 Methods

The proposed framework consists of three stages (Figure 1).

Encoder. The goal of an encoder $\mathcal{E}_\theta$ is to produce a preconditioner-generating representation of the physical gauge configuration U that determines the target Dirac equation to be solved. The encoder is parameterized by θ and maps U to the preconditioner-generating representation

$$\mathcal{E}_\theta(U) = \{V^{(1)}, \dots, V^{(N_h)}\}, \quad (2)$$

where each of the N_h heads h carries a set of matrix-valued lattice variables $V^{(h)} = \{\tilde{U}^{(h)}, W^{(h)}\}$.

Representation. $\tilde{U}^{(h)}$ is a set of link matrices $\{\tilde{U}_\mu^{(h)}(x) \in \mathbb{C}^{3 \times 3}\}_{x \in \Lambda, \mu=1, \dots, 4}$, and $W^{(h)}$ is a set of site matrices $\{W_c^{(h)}(x) \in \mathbb{C}^{3 \times 3}\}_{x \in \Lambda, c=1, \dots, N_c}$. These are exactly the objects the standard lattice operations (e.g., parallel transport) consume, so the decoder can be assembled from them without ever forming a matrix. We require neither linearity nor $SU(3)$ representations since linearity is required only of the preconditioner (which the decoder fully controls) and the lattice operators of the decoder do not require unitary inputs.

Decoder. The role of a decoder $\mathcal{M}_\phi$ in the proposed framework is to construct linear operators P_{ϕ_h} from the representations $V^{(h)}$, which are the building blocks of M^{-1} :

$$M^{-1} = \mathcal{H}\left(P_{\phi_1}[V^{(1)}], \dots, P_{\phi_{N_h}}[V^{(N_h)}]\right), \quad (3)$$

where $P_{\phi_1}[V^{(1)}], \dots, P_{\phi_{N_h}}[V^{(N_h)}] = \mathcal{M}_\phi(V^{(1)}, \dots, V^{(N_h)})$ and $\mathcal{H}$ is a fixed (not learned) function chosen so that M^{-1} is HPD. The (possibly empty) learnable parameters are $\phi = (\phi_1, \dots, \phi_{N_h})$, where ϕ_h is used only by P_{ϕ_h} and hence only on head h .

The encoder and decoder are jointly trained unsupervised using the unrolled PCG objective [Nytke et al., 2025],

$$\psi_K(\theta, \phi) = \text{PCG}\left(A, b; M_{\theta, \phi}^{-1}[U], K\right), \quad r_K(\theta, \phi) = b - A\psi_K(\theta, \phi), \quad \mathcal{L}(\theta, \phi) = \frac{\|r_K(\theta, \phi)\|_2}{\|b\|_2}, \quad (4)$$

where PCG is run for K steps (a hyperparameter), $\mathcal{L}$ is the loss, r_K is the PCG residual after K steps, and A and b define the linear system in Eq. (1).

For simplicity, we drop the head index when $N_h = 1$, writing $\mathcal{E}_\theta(U) = V$ and $M^{-1} = \mathcal{H}(P_\phi[V])$.

4 Numerical Experiments

We use $3+1$ dimensional hypercubic lattices of size L^4 with lattice spacing set to unity for simplicity. Gauge configurations are generated from the Wilson plaquette action [Wilson, 1974, Gattringer and Lang, 2009] using a heatbath algorithm implemented in `GaugeFields.jl` [Nagai and Tomiya, 2024], with the coupling $\beta = 6.0$. For the Dirac operator, we set the hopping parameter $\kappa = 0.155$, near the critical value $\kappa_c = 0.158$ for this lattice size and coupling. We train and evaluate the proposed framework with different encoder-decoder pairs on $L = 8$ lattices, and test zero-shot transfer by evaluating models trained exclusively on $L = 8$ directly on $L = 16$.

We compare four choices for the encoder $\mathcal{E}_\theta$. The **identity map**, $V = \mathcal{E}_\theta(U) = \{U\}$ (a singleton), passes the physical links directly to the decoder and serves to isolate its effects. The **CNN** of Sun et al. [2026], extended here to $3+1$ dimensions and $SU(3)$, is a multi-head ($N_h = 3$) fully convolutional network that produces $V^{(h)} = \{\tilde{U}^{(h)}\}$. The remaining two, **L-CNN** and **site-L-CNN**, are variants of the lattice gauge-equivariant CNN of Favoni et al. [2022], which constructs gauge-equivariant

Table 1: PCG iterations (speedup) at $\beta = 6.0$, $\kappa = 0.155$, averaged over 400 solves from 20 gauge configurations and 20 right-hand sides each. All models except AMG_{16} are trained or tuned on $L = 8$; AMG_{16} is re-tuned on $L = 16$. Speedup is relative to unpreconditioned CG. Lower iterations and higher speedup are better. The lower block lists encoder $\rightarrow$ decoder pairs built with our framework. Of these, $\text{CNN} \rightarrow \text{WNO}$ is the preconditioner of Sun et al. [2026] extended to $\text{SU}(3)$ and $3+1$ dimensions, and $\text{Identity} \rightarrow \text{PTLN}$ corresponds most closely to the model of Pfahler et al. [2026]; the other three pairs are new combinations enabled by the framework.

Encoder $\rightarrow$ Decoder	$L = 8$	$L = 16$
None (Unpreconditioned)	436 (1.0 $\times$)	1331 (1.0 $\times$)
AMG_8	117 (3.7 $\times$)	409 (3.3 $\times$)
AMG_{16}	—	264 (5.0 $\times$)
L-CNN $\rightarrow$ PTLN	56 (7.8 $\times$)	165 (8.1 $\times$)
Identity $\rightarrow$ PTLN	56 (7.8 $\times$)	168 (7.9 $\times$)
Site-L-CNN $\rightarrow$ CM-PTLN	57 (7.6 $\times$)	170 (7.8 $\times$)
CNN $\rightarrow$ HMP	277 (1.6 $\times$)	841 (1.6 $\times$)
CNN $\rightarrow$ WNO	278 (1.6 $\times$)	841 (1.6 $\times$)

features from untraced Wilson loops and parallel-transport convolutions; the L-CNN is a single-head model that produces link variables $V = \{\tilde{U}\}$ with each $\tilde{U}_\mu(x) \in \text{SU}(3)$, whereas the site-L-CNN (also $N_h = 1$) outputs each of its layer’s hidden activations, giving $V = \{W\}$ (see Appendix A).

We evaluate four decoder architectures $\mathcal{M}_\phi$. The **WNO** decoder is parameter-free, using $P[V^{(h)}] = (D^\dagger D)[V^{(h)}]$, the Wilson-Dirac normal operator built from head h ; setting $N_h = 1$ recovers the decoder used in Sun et al. [2026], but we find best results with $N_h = 3$. The **HMP** decoder uses a polynomial in the hopping operator $H[V^{(h)}]$ [Gattringer and Lang, 2009, Sec. 5.3.1], $f(H[V^{(h)}])$, fed into $P_{\phi_h}[V^{(h)}] = f^\dagger f$. Both decoders use a palindromic composition of $(P[V^{(h)}], 1 \leq h \leq N_h)$ for $\mathcal{H}$. The remaining two decoders both use $N_h = 1$ and admit exact adjoints which we use in $\mathcal{H}(P_\phi[V]) = P_\phi[V]^\dagger P_\phi[V]$. The **PTLN** [Pfahler et al., 2026] uses V -dependent parallel-transport layers with prescribed paths and learnable linear layers; we fix the path set at $L = 8$ and reuse it unchanged at $L = 16$ so nothing changes between lattice sizes. Finally, the **CM-PTLN** adds color mixing to the PTLN’s linear layers using the site-L-CNN’s final-layer output $\{W_c^{(N_\epsilon)}(x)\}$. See Appendix B for more decoder construction details.

We train five combinations of the listed encoders and decoders using the objective in Eq. (4), setting $K = 20$. Implementation details, model hyperparameters, and training set up are in Appendix C. We evaluate preconditioners by the number of PCG iterations required for convergence, averaged over a set of gauge configurations from held-out $L^4 = 8^4$ data and new $L^4 = 16^4$ data.

Table 1 summarizes the results. The best learned preconditioners require fewer iterations than both unpreconditioned CG and AMG in this small-volume setting. Their relative gains are preserved under zero-shot transfer from $L = 8$ to $L = 16$, with approximately $8\times$ reductions in iteration counts at both lattice sizes. In contrast, AMG’s iteration reduction drops from $3.7\times$ to $3.3\times$ when transferred without modification, and re-tuning at $L = 16$ raises it to $5.0\times$. A complete assessment of computational efficiency would also compare wall-clock solve times and floating-point operation counts, including both setup and solve costs. Limitations of our current implementation prevent a fair comparison of these measures (see Appendix C.6). Nevertheless, the low setup cost of the learned preconditioners may offer an advantage over AMG when solving systems across many gauge configurations, each of which requires a new preconditioner.

The most effective preconditioner on both $L = 8$ and $L = 16$ in terms of iterations is L-CNN $\rightarrow$ PTLN, although its improvement over Identity $\rightarrow$ PTLN is small. Both require 56 iterations at $L = 8$, while at $L = 16$ they require 165 and 168 iterations, respectively. Inspection of the learned representations suggests why this margin is so narrow: During training, the L-CNN encoder converges close to the identity map, i.e. $\text{L-CNN}(U) \approx U$ and the L-CNN $\rightarrow$ PTLN effectively becomes Identity $\rightarrow$ PTLN (see Appendix D).

We hypothesize that this is due to the shared structure of the PTLN and the true inverse $(D^\dagger D)[U]^{-1}$. In Appendix E, we show that for any choice of weights, the PTLN is a sum over lattice paths of parallel transporters along V , each multiplied by a learned matrix acting only on the spinor indices,

and that the hopping expansion of $(D^\dagger D)[U]^{-1}$ has the same form with transporters along U . For $V = U$, the two thus share the same color structure on every common path, so approximating the inverse reduces to learning the spinor weights, whereas a deviation $\tilde{U} \neq U$ alters the transporters in a way the spinor weights cannot undo. Since this holds for any weights, including at initialization, it is consistent with the rapid decay in Figure D.1; however, it only shows that $\tilde{U} = U$ is a possible good solution, not that it is the unique one. Verifying this hypothesis, and identifying regimes where a learned encoder helps (e.g., larger volumes or lighter quark masses), is left for future work.

5 Conclusion

We present an encoder–representation–decoder framework for training neural preconditioners for the Wilson-Dirac operator. Trained with an unsupervised unrolled PCG residual objective, the encoder takes the lattice gauge configuration as input and produces a representation that is used by the decoder to construct matrix-free preconditioners. Across five encoder–decoder pairs with different architectures, we show that the proposed framework reduces the number of PCG iterations required for convergence and transfers zero-shot to a larger lattice size without additional training. Future work will explore additional encoder–decoder architectures, investigate the learned preconditioners through spectral analysis, and evaluate the framework at production-scale lattice sizes.

Acknowledgments

We gratefully acknowledge use of the Swing cluster in the Laboratory Computing Resource Center at Argonne National Laboratory. This work is supported by the U.S. National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, <http://iaifi.org/>).

PES and WD were supported in part by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics, under Contract No. DE-SC0011090, and by the U.S. Department of Energy SciDAC5 award DE-SC0023116. PES was additionally supported by Early Career Award DE-SC0021006 and Simons Foundation Grant 994314 (Simons Collaboration on Confinement and QCD Strings), and thanks the Institute for Nuclear Theory at the University of Washington for its kind hospitality and stimulating research environment. This research was supported in part by the INT’s U.S. Department of Energy Grant No. DE-FG02-00ER41132.

The work of AS was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships (SULI) Program.

YS, SE, XS, and YL were supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program, under Contract DE-AC02-06CH11357 at Argonne National Laboratory (YS and SE) and DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory (XS and YL).

References

- James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/jax-ml/jax>.
- Achi Brandt and Oren E Livne. *Multigrid techniques*. Society for Industrial and Applied Mathematics, January 2011.
- M Brezina, R Falgout, S MacLachlan, T Manteuffel, S McCormick, and J Ruge. Adaptive algebraic multigrid. *SIAM J. Sci. Comput.*, 27(4):1261–1286, January 2006.
- Salvatore Calì, Daniel C. Hackett, Yin Lin, Phiala E. Shanahan, and Brian Xiao. Neural-network preconditioners for solving the Dirac equation in lattice gauge theory. *Phys. Rev. D*, 107:034508, 2023. doi: 10.1103/PhysRevD.107.034508.

- M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. Solving lattice QCD systems of equations using mixed precision solvers on GPUs. *Computer Physics Communications*, 181(9): 1517–1528, 2010. doi: 10.1016/j.cpc.2010.05.002.
- M. A. Clark, Bálint Joó, Alexei Strelchenko, Michael Cheng, Arjun Gambhir, and Richard Brower. Accelerating lattice QCD multigrid on GPUs using fine-grained parallelization, 2016.
- DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL <http://github.com/google-deepmind>.
- Matteo Favoni, Andreas Ipp, David I. Müller, and Daniel Schuh. Lattice gauge equivariant convolutional neural networks. *Phys. Rev. Lett.*, 128:032003, 2022. doi: 10.1103/PhysRevLett.128.032003.
- A Frommer, K Kahl, S Krieg, B Leder, and M Rottmann. Adaptive aggregation-based domain decomposition multigrid for the lattice Wilson–Dirac operator. *SIAM J. Sci. Comput.*, 36(4): A1581–A1608, January 2014.
- Jean H. Gallier. Basics of classical lie groups: The exponential map, lie groups, and lie algebras. In *Geometric Methods and Applications*, volume 38 of *Texts in Applied Mathematics*, pages 367–414. Springer, 2001. doi: 10.1007/978-1-4613-0137-0_14.
- Christof Gattringer and Christian B Lang. *Quantum chromodynamics on the lattice*. Lecture Notes in Physics. Springer, Berlin, Germany, 2010 edition, October 2009.
- Paul Häusner, Ozan Öktem, and Jens Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method, 2024.
- Bálint Joó, Chulwoo Jung, Norman H. Christ, William Detmold, Robert G. Edwards, Martin Savage, and Phiala Shanahan. Status and future perspectives for lattice gauge theory calculations to the exascale and beyond. *The European Physical Journal A*, 55(11), 2019. doi: 10.1140/epja/i2019-12919-7.
- Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.
- Christoph Lehner and Tilo Wettig. Gauge-equivariant neural networks as preconditioners in lattice QCD. *Phys. Rev. D*, 108:034503, 2023. doi: 10.1103/PhysRevD.108.034503.
- Yuki Nagai and Akio Tomiya. Juliaqcd: Portable lattice qcd package in julia language, 2024. URL <https://arxiv.org/abs/2409.03030>.
- Nicolas Nytko, Ali Taghibakhshi, Tareq Uz Zaman, Scott MacLachlan, Luke N. Olson, and Matthew West. Optimized sparse matrix operations for reverse mode automatic differentiation. *SIAM J. Sci. Comput.*, 47(5):C1115–C1143, 2025.
- Simon Pfahler, Daniel Knüttel, Christoph Lehner, and Tilo Wettig. A novel gauge-equivariant neural network architecture for preconditioners in lattice QCD. In *Proceedings of The 42nd International Symposium on Lattice Field Theory — PoS(LATTICE2025)*, page 027. SISSA Medialab, 2026. doi: 10.22323/1.518.0027.
- Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2 edition, 2003.
- Johannes Sappl, Laurent Seiler, Matthias Harders, and Wolfgang Rauch. Deep learning of preconditioners for conjugate gradient solvers in urban water related problems, 2019.

Yixuan Sun, Srinivas Eswar, Yin Lin, William Detmold, Phiala Shanahan, Yang Liu, Xiaoye Li, and Prasanna Balaprakash. Matrix-free neural preconditioner for the dirac operator in lattice gauge theory. *ACM Trans. AI Sci.*, August 2026. doi: 10.1145/3841473. URL <https://doi.org/10.1145/3841473>. Just Accepted.

Akio Tomiya, Hiroshi Ohno, and Yuki Nagai. CASK: A gauge covariant transformer for lattice gauge theory. In *Proceedings of The 41st International Symposium on Lattice Field Theory — PoS(LATTICE2024)*, page 030, Trieste, Italy, January 2025. Sissa Medialab.

Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Academic Press, San Diego, CA, November 2000.

Kenneth G. Wilson. Confinement of quarks. *Phys. Rev. D*, 10:2445–2459, 1974. doi: 10.1103/PhysRevD.10.2445.

A L-CNN Architecture and Variants

A.1 Changes to the Original L-CNN

A lattice gauge-equivariant CNN (L-CNN) [Favoni et al., 2022] builds gauge equivariance into the architecture rather than learning it. It operates on a pair (U, W) , where $U = \{U_\mu(x) \in \text{SU}(3)\}$ are link variables and $W = \{W_c(x) \in \mathbb{C}^{3 \times 3}\}$ are $N_c = N_{\text{ch}}$ channels of matrix-valued site variables. The channels are initialized with untraced Wilson loops built from the links and refined by a stack of gauge-equivariant blocks that parallel transport them between neighboring sites and take learned products of them. These can form Wilson loops of arbitrary shape and size and make L-CNNs universal approximators of gauge-equivariant functions on the lattice [Favoni et al., 2022].

Our use of the L-CNN differs from the original in two respects:

1. **The output is a gauge field, not a scalar.** Favoni et al. [2022] terminate their networks with a trace layer, producing scalars for each site that a conventional CNN then regresses onto a physical observable. We do not use a trace layer, but instead terminate with their (proposed, but unused) L-Exp layer,

$$\tilde{U}_\mu(x) = \exp\left(i \sum_c \beta_{\mu,c} [W_c(x)]_{\text{ah}}\right) U_\mu(x), \quad (\text{A.1})$$

where $[\cdot]_{\text{ah}}$ is the traceless anti-Hermitian part and the $\beta_{\mu,c} \in \mathbb{R}$ are learned. The exponent lies in $\mathfrak{su}(3)$, so $\tilde{U}_\mu(x) \in \text{SU}(3)$ is guaranteed [Gallier, 2001, Theorem 14.4.2]. The whole model is a gauge-equivariant map from $\text{SU}(3)$ links to $\text{SU}(3)$ links, with a single head ($N_h = 1$), so its representation is $V = \{\tilde{U}\}$ and it is compatible with all of our decoders (Appendix B). We initialize $\beta_{\mu,c} \sim \mathcal{N}(0, 0.1^2)$, a small perturbation of the identity map ($\beta_{\mu,c} = 0 \ \forall \mu, c$ reproduces $\tilde{U} = U$).

2. **A gauge-equivariant normalization layer.** We insert a per-channel RMS normalization after each block,

$$W_c = \frac{W_c}{\sqrt{\text{mean}_x \|W_c(x)\|_F^2 + \varepsilon}} \cdot \gamma_c, \quad (\text{A.2})$$

where $\gamma_c \in \mathbb{R}$ is a learnable scalar and $\|\cdot\|_F$ is the Frobenius norm. The Frobenius norm is gauge invariant [Tomiya et al., 2025], so rescaling by it preserves equivariance. This stabilizes training and prevents the exploding activations caused by the repeated products that build up the channels.

A.2 Site-L-CNN

The **site-L-CNN** is identical to the L-CNN in Appendix A.1, except that it omits the L-Exp layer and outputs the site variables from every layer. It has a single head ($N_h = 1$), and its output representation is

$$V = \{W\} = \{W_c(x)\}_{x \in \Lambda, 1 \leq c \leq N_c}, \quad W_c(x) \in \mathbb{C}^{3 \times 3}, \quad (\text{A.3})$$

where $N_c = N_{\text{ch}} \cdot N_\ell$ is the number of channels per layer times the number of layers. These are the site variables consumed by the CM-PTLN decoder (Appendix B).



Figure A.1: The L-CNN architecture used with added L-Exp and RMSnorm layers.

B Decoder construction

Here we go into more detail on some of the decoders we use.

Wilson Normal Operator (WNO). This parameter-free decoder uses $P[V^{(h)}] = (D^\dagger D)[V^{(h)}]$ (the Wilson normal operator) and

$$M^{-1} = \mathcal{H}(P[V^{(1)}], \dots, P[V^{(N_h)}]) = P[V^{(1)}] \cdot P[V^{(2)}] \dots P[V^{(N_h)}] \dots P[V^{(2)}] \cdot P[V^{(1)}], \quad (\text{B.1})$$

which is HPD by construction. When using link variable representations $\tilde{U}^{(h)}$ and $N_h = 1$, this recovers the decoder of Sun et al. [2026].

Hopping Matrix Polynomial (HMP). We can write the Wilson-Dirac operator as $D[V] = 1 - \kappa H[V]$, where $H[V]$ is the *hopping matrix* [Gattringer and Lang, 2009, Section 5.3.1] and κ is the hopping parameter. The Neumann series $D[V]^{-1} = \sum_{j \geq 0} \kappa^j H[V]^j$ then converges for $\kappa < 1/8$ [Gattringer and Lang, 2009, Eq. (5.58)], which our value of $\kappa = 0.155$ falls outside of. This suggests instead trying a *learnable, truncated* polynomial in the hopping matrix,

$$f(V^{(h)}) = \sum_{j=0}^n a_{h,j} \kappa^j H[V^{(h)}]^j, \quad P_{\phi_h}[V^{(h)}] = f(V^{(h)})^\dagger f(V^{(h)}), \quad (\text{B.2})$$

$$\phi_h = \{a_{h,j}\}_{j=0}^n \subset \mathbb{R} \text{ learnable},$$

using the same $\mathcal{H}$ operator as the WNO decoder (Eq. B.1), making the preconditioner HPD by construction. We find best results with $n = 1$.

Parallel-Transport Linear Network (PTLN). The PTLN [Pfahler et al., 2026] is built from two layer types: a **parallel transport layer (PT)** applies a chain of single hops along a fixed (hardcoded) path to each of its input fields, acting on the color indices, and a traditional **linear layer (L)** then transforms the spinor indices; the linear layer weights are the only trainable parameters. Straying from Pfahler et al. [2026], we construct the exact adjoint $P_\phi^\dagger$ by reversing the order of the layers, conjugate-transposing each linear layer weight matrix, and reversing and flipping each parallel transport path; we then use $\mathcal{H}(P_\phi[V]) = P_\phi[V]^\dagger P_\phi[V]$, which is HPD by construction. Importantly, we have the PT layers transport along the representation $V = \mathcal{E}_\theta(U)$ rather than along the physical links U .

Color-Mixing PTLN (CM-PTLN). This model replaces each PTLN linear layer weight S with a sum of Kronecker products $\sum_c S_c \otimes W_c^{(N_\ell)}(x)$, in which S_c acts on the spinor indices as before while the site-L-CNN’s final-layer output $W_c^{(N_\ell)}(x)$ acts on the color indices. This lets the linear layers mix the three colors at a site, something the PTLN’s linear layers cannot do.

C Model hyperparameters and training details

C.1 Implementation

The framework is implemented in JAX with Equinox modules and Optax optimizers [Bradbury et al., 2018, Kidger and Garcia, 2021, DeepMind et al., 2020]. All lattice operators (the Wilson-Dirac operator $D[U]$, the hopping matrix $H[U]$, parallel transport, and the L-CNN layers) are written as matrix-free matrix vector products, so no preconditioner is ever assembled. Boundary conditions are periodic in the three spatial directions and antiperiodic in time. Training runs data-parallel on a single node with eight Nvidia A100 GPUs.

C.2 Data

Gauge configurations are generated with the heatbath algorithm of `Gaugefields.jl` [Nagai and Tomiya, 2024] from the Wilson plaquette action [Wilson, 1974, Gattringer and Lang, 2009] at $\beta = 6.0$, using independent Markov chains run in parallel, 100 thermalization sweeps per chain, and 10 sweeps discarded between saved configurations. The $L = 8$ ensemble is split 80/20 into training and validation sets by a fixed random permutation. The $L = 16$ ensemble is generated by the same procedure and is used only for zero-shot evaluation; no model sees $L = 16$ data during training. The hopping parameter is $\kappa = 0.155$ for both training and evaluation.

C.3 Model Hyperparameters

The hyperparameters used for the encoders and decoders are listed in Tables C.1 and C.2, respectively.

Table C.1: Hyperparameters used for encoders.

Encoder	Hyperparameters
Identity	None (no parameters)
CNN	4 complex 4D convolutions, kernel 3^4 , hidden width 16, PReLU; 36 input and output channels (4 directions $\times 3 \times 3$ color entries); a 3-layer shared backbone feeding 3 output heads, each producing its own $\tilde{U}^{(h)}$
L-CNN	3 (L-Conv + L-Bilin) blocks, each with 8 channels, kernel size 2 (single hop); gauge-equivariant RMS norm after each block ($\varepsilon = 10^{-6}$); channels seeded with the 6 plaquettes and 4 Polyakov loops; L-Exp read-out with $\beta_{\mu,c}$ initialized to $\mathcal{N}(0, 0.1^2)$
Site-L-CNN	Same as the L-CNN but without the L-Exp read-out

Table C.2: Hyperparameters used for decoders.

Decoder	Hyperparameters
WNO	$(D^\dagger D)[V]$; no parameters of its own
HMP	We use a polynomial of degree $n = 1$, so two learnable real coefficients a_0, a_1 per head
PTLN	3 PT layers with 25 paths each (the identity plus shifts of $\pm 1, \pm 2, \pm 4$ in each of the 4 directions) giving 2 hidden linear blocks and layer widths $1 \rightarrow 25 \rightarrow 25 \rightarrow 25 \rightarrow 1$; each linear layer acts as a complex 4×4 matrix on the spinor indices; weights initialized as $\mathcal{N}(0, (10^{-3})^2)$; a channel-0 residual skip follows every linear layer, including the output layer
CM-PTLN	$N_{\text{ch}} = 8$ in the color-mixing linear layer, matching the number of channels in the site-L-CNN; otherwise identical to the PTLN

C.4 Training

All five pairs are trained with the unrolled PCG objective of Eq. (4) with $K = 20$ steps. At every optimizer step we draw 8 fresh complex Gaussian right-hand sides per gauge configuration, run K steps of PCG, and average $\|r_K\|/\|b\|$ over right-hand sides and over the batch. No right-hand sides are reused between epochs.

The validation loss is the same unrolled-PCG residual evaluated on the held-out split, computed once per epoch. Training stops when it fails to improve on the best value by at least 1% for patience consecutive epochs, and the checkpoint with the lowest validation loss is the one evaluated. The epoch cap is never reached: every run early-stops within 20–60 epochs, or roughly 30 minutes to a few hours of wall time.

Table C.3: Hyperparameters used for training.

Setting	Value
Optimizer	Adam
Peak learning rate	10^{-3}
Schedule	Cosine decay over 10,000 steps
Batch size	32 gauge configurations (4 per GPU)
Unrolled PCG steps K	20
Right-hand sides per step	8
Early-stopping patience	10 epochs

C.5 Evaluation

Test-time solves use PCG on $A = (D^\dagger D)[U]$ with $b = D^\dagger[U] \eta$ for complex Gaussian sources η , run in double precision to a relative residual tolerance of $\|r\|/\|b\| < 10^{-8}$ with no iteration cap; every solve reported in Table 1 converged. The encoder forward pass producing V is run once per gauge configuration and reused across all right-hand sides for that configuration.

C.6 Cost Comparison

To complement the iteration counts, we report floating-point operation counts (FLOPs) and wall-clock times *per application* for the current PCG implementations. Table C.4 reports the cost of applying the normal operator $A = D^\dagger D$ for unpreconditioned CG or the preconditioner for PCG. Table C.5 reports setup FLOPs per gauge configuration. The neural methods use compiled JAX, whereas AMG and an additional unpreconditioned CG control use C++/QUDA. Application times are medians for both backends.

Table C.4: FLOPs and median wall-clock time *per application* in the PCG benchmark. For unpreconditioned CG, application cost refers to A ; otherwise, it refers to the preconditioner alone M^{-1} .

Method	Backend	GFLOPs	Time (ms)
Unpreconditioned CG	JAX	0.011207	0.7519
Unpreconditioned CG	QUDA	0.011207	0.0391
AMG	QUDA	0.277381	0.9429
Identity $\rightarrow$ PTLN	JAX	4.446065	8.4520
L-CNN $\rightarrow$ PTLN	JAX	4.446065	8.4457
Site-L-CNN $\rightarrow$ CM-PTLN	JAX	21.366970	11.4374
CNN $\rightarrow$ HMP	JAX	0.219955	2.8430
CNN $\rightarrow$ WNO	JAX	0.217006	2.9436

The unpreconditioned controls illustrate the effect of implementation on wall-clock performance. Applying the same normal operator requires 0.011207 GFLOPs in both backends, yet takes 0.7519 ms in JAX and 0.0391 ms in QUDA, a difference of approximately $19\times$. Within QUDA, applying the current matrix-free AMG preconditioner requires 0.277381 GFLOPs and 0.9429 ms, substantially more than an application of A . Our AMG implementation is adapted for use with PCG on the normal equations, and this adaptation may introduce additional overhead. Meanwhile, the learned preconditioners, not yet optimized for speed, have higher per-application costs than the normal operator in JAX. These backend differences and implementation costs prevent a fair comparison of the preconditioning methods’ computational efficiency in the current setup.

Despite these limitations, setup costs suggest a potential advantage of learned preconditioners when solving systems across many gauge configurations. AMG requires approximately 483.3 GFLOPs of setup per configuration, compared with 0–12.1 GFLOPs for the five neural methods. Setup is performed once per configuration and reused across its right-hand sides, making lower setup costs particularly relevant when relatively few right-hand sides are solved per configuration. As immediate future work, we will compare optimized implementations using consistent timing statistics and measurements of both setup and solve times to assess the resulting end-to-end performance.

Table C.5: Setup FLOPs per gauge configuration for the PCG benchmark. Setup is performed once for *each new gauge configuration* and reused across all right-hand sides on that configuration.

Method	Backend	Setup (GFLOPs)
Unpreconditioned CG	JAX / QUDA	0
AMG	QUDA	483.257319
Identity $\rightarrow$ PTLN	JAX	0
L-CNN $\rightarrow$ PTLN	JAX	12.116460
Site-L-CNN $\rightarrow$ CM-PTLN	JAX	3.665756
CNN $\rightarrow$ HMP	JAX	1.474560
CNN $\rightarrow$ WNO	JAX	1.474560

D Analysis of trained L-CNN

We find that during training of the L-CNN $\rightarrow$ PTLN model, extra gauge-equivariant capacity (the L-CNN) added to the model (the PTLN) is discarded in favor of an effectively simpler architecture. Although the L-CNN is not initialized to the identity mapping, it converges during training to $L\text{-CNN}(U) \approx U$. We track this *deviation from identity* as

$$\text{deviation}(\tilde{U}) = \text{mean}_{x, \mu} \frac{\|\tilde{U}_\mu(x) - U_\mu(x)\|_F}{\|U_\mu(x)\|_F}. \quad (\text{D.1})$$

If $\tilde{U}_\mu(x) \approx U_\mu(x)$ on average, then deviation ≈ 0 . From Figure D.1, we see that the L-CNN $\rightarrow$ PTLN model becomes essentially equivalent to the simpler Identity $\rightarrow$ PTLN model.

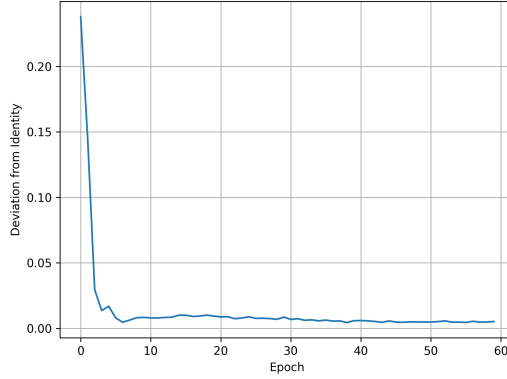


Figure D.1: Deviation from identity during training for an L-CNN encoder paired with a PTLN decoder. The L-CNN converges close to the identity mapping, showing that extra gauge-equivariant capacity is discarded in favor of a simpler architecture.

E The PTLN as a learned hopping expansion

In this section, we show that the PTLN [Pfahler et al., 2026] has the same structure as the hopping expansion of $D[V]^{-1}$ [Gattringer and Lang, 2009, Section 5.3.1].

Writing $\tilde{U}_\mu(x)$ for the link variables of V , we define the forward and backward hops along V in direction $\mu \in \{1, \dots, 4\}$ as

$$(T_{+\mu}\psi)(x) = \tilde{U}_\mu(x) \psi(x + \hat{\mu}), \quad (T_{-\mu}\psi)(x) = \tilde{U}_\mu(x - \hat{\mu})^\dagger \psi(x - \hat{\mu}), \quad (\text{E.1})$$

with $\tilde{U}_\mu(x)$ acting on the color indices of the quark field ψ and the antiperiodic temporal boundary condition absorbed into the links. The parallel transport along a path $p = (p_1, \dots, p_n)$ of hops $p_i \in \{\pm 1, \dots, \pm 4\}$ is then $T_p = T_{p_1} \cdots T_{p_n}$ [Gattringer and Lang, 2009, Eq. (5.61)], and we have $T_p T_{p'} = T_{pp'}$ for concatenated paths.

A PTLN alternates N PT layers with $N + 1$ linear layers. We write linear layer k as a block matrix $S^{(k)}$, where $S_{ba}^{(k)} \in \mathbb{C}^{4 \times 4}$ is the weight from channel a to channel b ; PT layer k , which transports each of its channels a along a path $p_a^{(k)}$, we write as the diagonal matrix $T^{(k)} = \text{diag}(T_{p_1^{(k)}}, T_{p_2^{(k)}}, \dots)$. Note that $S^{(0)}$ is a column of blocks and $S^{(N)}$ is a row of blocks since the PTLN maps a single field to a single field. The PTLN is then the product

$$P_\phi[V] = S^{(N)} T^{(N)} S^{(N-1)} T^{(N-1)} \dots S^{(1)} T^{(1)} S^{(0)}. \quad (\text{E.2})$$

The weights act only on the spinor indices and are identical at every site, and the transporters leave the spinor indices untouched, so $S T_p = T_p S$ for every weight S and path p . Multiplying out Eq. (E.2) gives one term per channel route $\mathbf{a} = (a_1, \dots, a_N)$, and commuting the weights to the left and concatenating the transports in each term gives

$$P_\phi[V] = \sum_{\mathbf{a}} S_{\mathbf{a}} T_{q(\mathbf{a})}[V], \quad (\text{E.3})$$

where $S_{\mathbf{a}} = S_{a_N}^{(N)} \dots S_{a_2 a_1}^{(1)} S_{a_1}^{(0)}$ is the product of weights along the route and $q(\mathbf{a}) = p_{a_N}^{(N)} \dots p_{a_1}^{(1)}$ is its composite path. Distinct routes can produce the same composite path; e.g., if the PT layers contain the empty path (no hop), a path can be selected in any one of the layers with the empty path in the others. Grouping these terms yields

$$P_\phi[V] = \sum_{q \in \mathcal{Q}} S_q T_q[V], \quad S_q = \sum_{\mathbf{a}: q(\mathbf{a})=q} S_{\mathbf{a}}, \quad (\text{E.4})$$

where $\mathcal{Q} = \{q(\mathbf{a})\}$ is the set of all composite paths. Importantly, this decomposition holds for any choice of weights, including those at initialization.

The hopping matrix

$$H[V] = \sum_{\mu=1}^4 \left[(\mathbb{1} - \gamma_\mu) T_{+\mu} + (\mathbb{1} + \gamma_\mu) T_{-\mu} \right]$$

[Gattringer and Lang, 2009, Eq. (5.56)] is itself a sum of single hops with fixed spinor weights (the Euclidean gamma matrices γ_μ). Labeling the eight hops by $h \in \{\pm 1, \dots, \pm 4\}$, we write it compactly as

$$H[V] = \sum_h S_h^H T_h, \quad S_{\pm\mu}^H = \mathbb{1} \mp \gamma_\mu. \quad (\text{E.5})$$

The spinor weights S_h^H commute with the transports by the same argument as for the PTLN, so expanding the j -th power gives one term per sequence of hops $q = (h_1, \dots, h_j)$,

$$\begin{aligned} H[V]^j &= \sum_{h_1, \dots, h_j} S_{h_1}^H T_{h_1} \dots S_{h_j}^H T_{h_j} && [\text{Gattringer and Lang, 2009, Eq. (5.60)}] \\ &= \sum_{|q|=j} S_q^H T_q[V], \quad S_q^H = S_{h_1}^H \dots S_{h_j}^H, && (\text{E.6}) \end{aligned}$$

where $|q|$ is the number of hops in q . Each path q appears in exactly one power, $j = |q|$, so for $\kappa < 1/8$ the Neumann series becomes a single sum over all paths,

$$\begin{aligned} D[V]^{-1} &= \sum_{j \geq 0} \kappa^j H[V]^j && [\text{Gattringer and Lang, 2009, Eq. (5.58)}] \\ &= \sum_q \kappa^{|q|} S_q^H T_q[V]. && (\text{E.7}) \end{aligned}$$

Comparing Eqs. (E.4) and (E.7), the hopping expansion sums over all paths with fixed weights $\kappa^{|q|} S_q^H$, while the PTLN sums over the restricted set $\mathcal{Q}$ with learned weights S_q . Since we use $\kappa = 0.155 > 1/8$, the Neumann series may not converge, and the PTLN effectively replaces the standard coefficients with learned ones while retaining the path structure. Since the adjoint of a transport is the transport along the reversed path, the adjoint of a path sum is again a path sum, and multiplying two path sums, commuting the spinor weights to the left as before, yields a sum

over concatenated paths; therefore, the same correspondence holds between our HPD preconditioner $M^{-1} = P_\phi[V]^\dagger P_\phi[V]$ and its target $(D^\dagger D)[U]^{-1} = D[U]^{-1}(D[U]^{-1})^\dagger$. In addition, because the weights act only on spinor indices, all color information in M^{-1} comes from the transports $T_q[V]$, and hence from the representation V produced by the encoder. With the identity encoder, $\tilde{U} = U$, and these are the same transports that appear in the hopping expansion of $(D^\dagger D)[U]^{-1}$.