Learning Low Modes of the Wilson Dirac Operator with Gauge-Equivariant Networks

Lattice 2026 — Algorithms and artificial intelligence

Minjae Choi[1] Hiroshi Ohno[1]

[1]University of Tsukuba

July 26 – August 1, 2026
University of Maryland, College Park

筑波大学
University of Tsukuba

Outline

1. Motivation
2. Case study: multigrid
3. Gauge equivariance
4. Network architecture & training
5. MG test-vector comparison
6. Summary

Why learn low modes?

- Low Dirac modes are widely used in lattice QCD: deflation and multigrid, low-mode averaging, all-to-all propagators, disconnected diagrams, and spectral studies.
- Eigensolvers and MG setup must be repeated for every configuration; iterative solves also slow down near κ_c.
- A trained model can be readily applied to new configurations within the same ensemble.

Questions

1. Can a **gauge-equivariant** network generate useful approximations to the low modes of D?
2. Does it generalize to configurations from the same ensemble that were not used for training?

This talk: gauge-equivariant low-mode prediction

Approach

Train a network built from gauge-equivariant layers using a Rayleigh–Ritz objective for $D^\dagger D$.

What we use and test

1. Gauge-equivariant PT layers (Pfahler et al., arXiv:2602.23840)
2. Low-mode learning that generalizes across configurations within the same ensemble.
3. Outputs are usable as test vectors for two-level MG: approximate low modes for the coarse space.

Evaluation

We need a concrete way to evaluate whether the learned low modes are useful. In this work, we use multigrid as our case study.

Two-level multigrid: what and why

Problem

Iterative solves of $D[U]\,x = b$ slow down near κ_{crit} because $D^\dagger D$ has small eigenvalues.

Two-level multigrid (MG) idea

- Build a **prolongator** P from N_v test vectors $\{v_j\}_{j=1}^{N_v}$ that capture the low-mode subspace of $D^\dagger D$.
- Form the coarse operator $D_c = P^\dagger D P$.
- A V-**cycle** restricts the residual, solves on the coarse grid, and prolongs the correction; used as an FGMRES preconditioner.

Setup cost

For each gauge configuration, MG setup chooses $\{v_j\}$ and builds P, D_c.
Our test: replace $\{v_j\}$ with learned network outputs and skip the adaptive refinement loop.

Why gauge equivariance?

Local $\mathrm{SU}(N_c)$ gauge transformation $G(x)$:

$$U_\mu(x) \mapsto G(x)\, U_\mu(x)\, G^\dagger(x+\hat{\mu}), \qquad \psi(x) \mapsto G(x)\, \psi(x).$$

Let $H[U] \equiv D[U]^\dagger D[U]$. Gauge equivariance gives

$$H[U^G] = G\, H[U]\, G^\dagger.$$

Therefore,

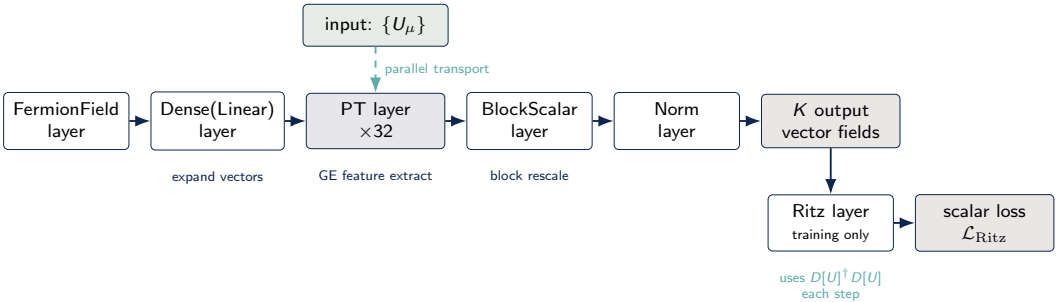
$$H[U]\psi_k = \lambda_k \psi_k \quad \Longrightarrow \quad H[U^G](G\psi_k) = \lambda_k (G\psi_k),$$

Meaning: λ_k is **gauge invariant**; ψ_k is **gauge covariant**: $H[U^G](G\psi_k) = \lambda_k(G\psi_k)$.

What does this mean for learning?

If every layer preserves the gauge-transformation law, the full network is gauge equivariant by construction. It does not need to learn gauge equivariance from the training data.

Architecture at a glance



Training and output

Each step samples one configuration: PT layers use $\{U_\mu\}$, Rayleigh–Ritz loss uses $D[U]^\dagger D[U]$. After training, K outputs span the learned low-mode subspace.

Ritz layer: Rayleigh–Ritz loss

Let X be the matrix whose columns are the K output vectors from the previous layer:

$$X = \begin{bmatrix} x_0 & \cdots & x_{K-1} \end{bmatrix} \in \mathbb{C}^{N \times K}.$$

Layer output

$$\boxed{\mathcal{L}_{\mathrm{Ritz}}(X) = \sum_{k=0}^{K-1} \lambda_k \in \mathbb{R}.}$$

The generalized Ritz values λ_k are obtained from

$$A = X^\dagger D[U]^\dagger D[U] X, \qquad B = X^\dagger X, \qquad A\mathbf{u}_k = \lambda_k B \mathbf{u}_k.$$

With $\tilde{x}_k = X\mathbf{u}_k$, each Ritz value is a Rayleigh quotient:

$$\lambda_k = \frac{\langle \tilde{x}_k | D[U]^\dagger D[U] | \tilde{x}_k \rangle}{\langle \tilde{x}_k | \tilde{x}_k \rangle}.$$

Therefore, the same loss can be written as

$$\boxed{\mathcal{L}_{\mathrm{Ritz}} = \sum_{k=0}^{K-1} \lambda_k = \mathrm{tr}(B^{-1}A) = \mathrm{tr}\left[(X^\dagger X)^{-1} X^\dagger D[U]^\dagger D[U] X \right].}$$

Practical advantage: Increasing K adds little training cost.

PT layer: parallel-transport convolution

Let $\mathcal{P}$ be a fixed set of lattice paths and $T_p[U]$ the parallel-transport operator along path p:

$$T_p[U] = H_{p_{n_p}} \cdots H_{p_2} H_{p_1}.$$

Input and output

$$\underbrace{\big(\Phi = (\phi_1, \ldots, \phi_n),\ U\big)}_{n \text{ input vectors} + \text{gauge field}} \xrightarrow{\text{PT layer}} \underbrace{\Psi = (\psi_1, \ldots, \psi_m)}_{m \text{ output vectors}}.$$

Layer operation

$$\boxed{\psi_a(x) = \sum_{b=1}^{n} W_{ab} \left(T_{p_b}[U]\phi_b\right)(x), \qquad a = 1, \ldots, m.}$$

Each input vector ϕ_b is paired with a path p_b. $T_{p_b}[U]$ transports ϕ_b to site x; the results are mixed by trainable spin matrices $W_{ab} \in \mathbb{C}^{4\times 4}$.

Gauge equivariance

$$\phi_b(x) \to G(x)\phi_b(x) \quad \Longrightarrow \quad \left(T_{p_b}[U]\phi_b\right)(x) \to G(x)\left(T_{p_b}[U]\phi_b\right)(x),$$

hence $\psi_a(x) \to G(x)\psi_a(x)$.

Adapted from Pfahler et al., arXiv:2602.23840.

Local layers: FermionField, Dense(Linear), BlockScalar

FermionField layer (trainable input fermion fields):

$$\Phi^{(0)} = (\phi_1^{(0)}, \ldots, \phi_n^{(0)}), \qquad \phi_b^{(0)}(x) \in \mathbb{C}^{4\times N_c}.$$

Every site, spin, and color component of $\phi_b^{(0)}$ is a **trainable parameter**.
Dense(Linear) layer ($n \to m$; PT mixing without hopping, $T_{p_b} = I$):

$$\chi_a(x) = \sum_{b=1}^{n} W_{ab}\, \phi_b(x), \qquad W_{ab} \in \mathbb{C}^{4\times 4}.$$

BlockScalar layer (blockwise scalar rescale): the lattice is partitioned into spatial blocks; each site x belongs to a block $B(x)$ of linear size r ($r^3 \times r_t$ in 4D):

$$\chi_k(x) = w_{B(x),k}\, \psi_k(x), \qquad w_{B,k} \in \mathbb{C}.$$

One trainable scalar per block and output mode k.

Block size r is a training hyperparameter; we use r=2 in our trained models.

Unlike the PT layer, these three local layers act only on fermion fields and do not take gauge links as input.

Note: fixed trainable FermionField inputs $\Rightarrow$ the full network is **not yet fully gauge equivariant**.

Training setup

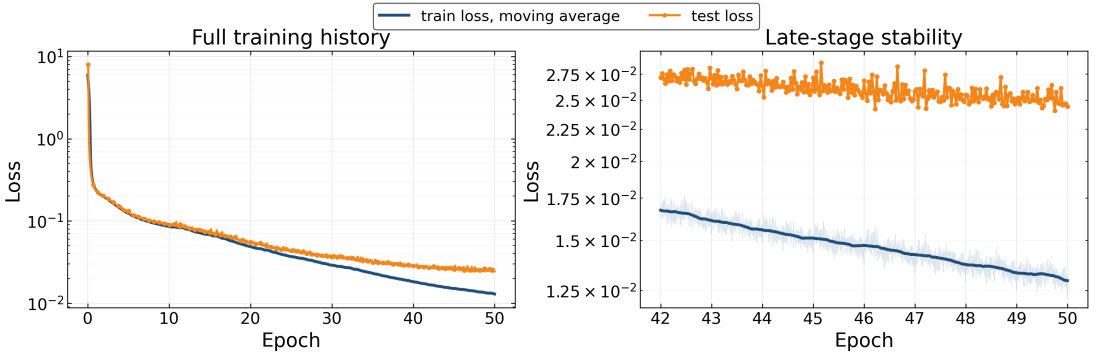
Learned low modes at $\kappa = 0.158$

Item	Setting
Gauge ensemble	quenched (heatbath), β=6.0
Lattice	$16^3 \times 32$
Dirac operator	Wilson, κ=0.158
Architecture	32PT stack; 33 paths (maxlen 5); K=6, 12, 24, 48, 64 outputs
Initialization	Xavier
Optimizer / loss	Adam; $\mathcal{L}_{\mathrm{Ritz}}$ on $D^\dagger D$
Training	50 epochs; 320 gauge configurations
Evaluation	20 configurations *not used for training*

Monitor $\mathcal{L}_{\mathrm{Ritz}}$ on configurations not used for training every epoch (next slide).
Four trained networks with $K{=}N_v \in \{12, 24, 48, 64\}$ enter the MG benchmark.

Training progress

Network training at $\kappa = 0.158$ (33 paths, $K = 6$ case)

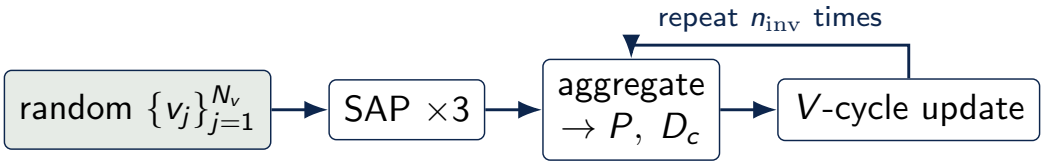


final train = 0.0129 final test = 0.0244 min test = 0.0241

50 epochs = 16,000 steps; about 3 days on NVIDIA A100 (Julia; Gaugefields.jl, LatticeDiracOperators.jl).

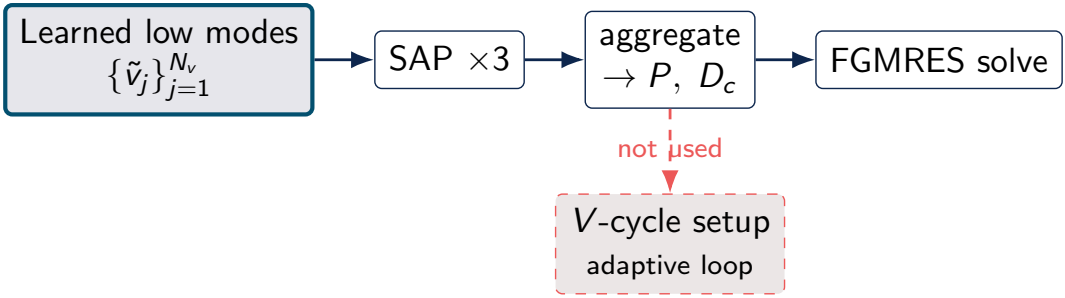
Classical setup: adaptive V-cycle

Classical $\mathrm{IDsetup}(n_{\mathrm{inv}})$ [Brannick et al., arXiv:1303.1377]



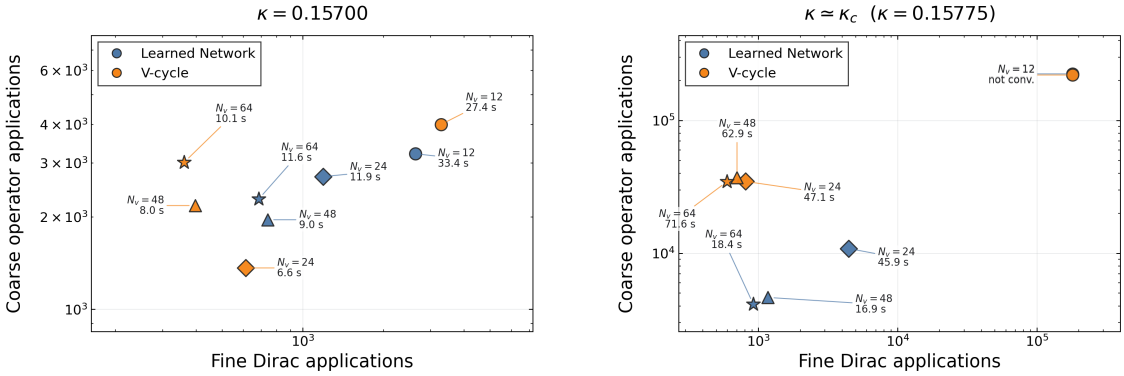
- **Test vectors**: approximate low-mode vectors used to build the prolongator P; N_v controls the size of the coarse problem used for preconditioning.
- Rebuild P, $D_c = P^\dagger D P$; update $v_j \leftarrow \mathcal{C}^{(\nu)} v_j$ on $Dv{=}0$.
- Done *per gauge configuration*.
- Classical setup used for comparison on the next slides.

Our low-mode test-vector setup



- Network outputs replace random $\{v_j\}_{j=1}^{N_v}$.
- We apply **only the first SAP$\times$3 smoothing step**, then aggregate to P and D_c.
- No adaptive V-cycle setup loop (n_{inv} iterations in the classical setup).
- Same solver stage as the classical setup; only the v_j *setup* differs.
- Benchmark: four trained networks ($K{=}N_v \in \{12, 24, 48, 64\}$; **4000 steps**), compared to the classical setup on configs A and B (not used for training; next slides).

Main result: learned low modes vs. V-cycle setup (config A)



Four trained networks ($K{=}N_v \in \{12, 24, 48, 64\}$; 4000 steps).

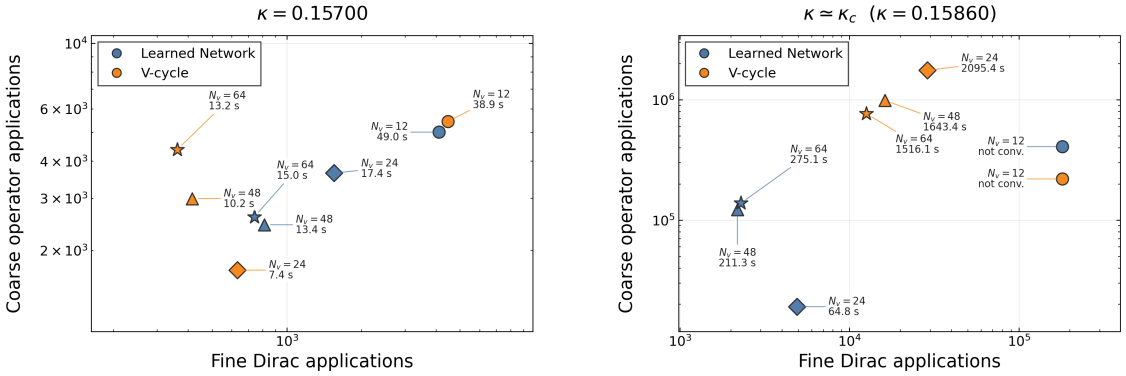
Left: $\kappa = 0.157$; right: near κ_c.

Blue: learned network vectors; orange: classical V-cycle setup.

Lower left is better.

Solve to residual 10^{-10}; axes: fine / coarse operator applications.

Main result: learned low modes vs. V-cycle setup (config B)



Same protocol on config B.

Left: $\kappa = 0.157$; right: near κ_c.

Blue: learned network vectors; orange: classical V-cycle setup.

Lower left is better.

Near κ_c, N_v=12 fails to converge for both.

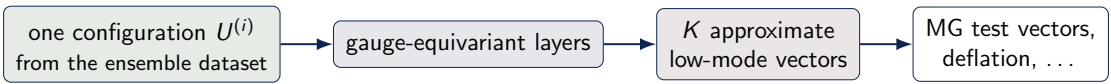
Summary

What learned low modes offer

- **Generalization:** observed across configurations within the same ensemble.
- **Two-level multigrid:** the network vectors replace the adaptive V-cycle setup loop.
- **Other Dirac operators:** the same framework can be trained with other lattice Dirac operators.
- **Next tests:** other ensembles, larger lattices, and other lattice QCD applications.

Thank you — questions welcome.

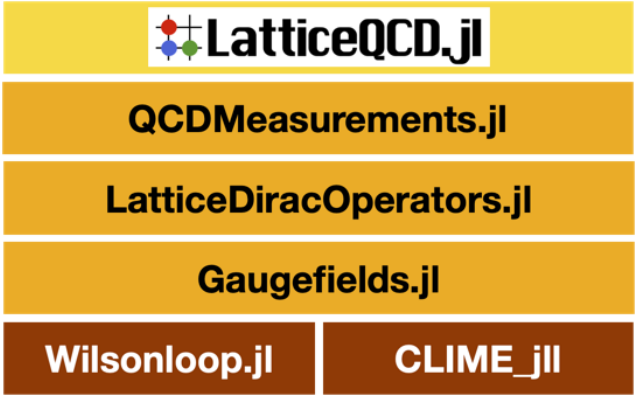
Backup: Network idea



- **Input:** gauge links $\{U_\mu\}$.
- **Output:** K vector fields whose span approximates the lowest subspace of $D^\dagger D$.
- Trained on the ensemble; applied to new configurations.

Backup: computational setup

- **GPU:** NVIDIA A100 80GB PCIe
- **Lattice code:** LatticeQCD.jl



Backup: Wilson Dirac operator

Wilson discretization with hopping parameter κ:

$$D(x,y) = \delta_{x,y} - \kappa \sum_{\mu=\pm 1}^{\pm 4} \left(1 \mp \gamma_\mu\right) U_\mu(x)\, \delta_{x+\hat{\mu},y}.$$

- Sparse, non-Hermitian; γ_5-Hermiticity: $\gamma_5 D \gamma_5 = D^\dagger$.
- Small eigenvalues near κ_{crit} slow Krylov solvers.
- Reference training: κ=0.158 on heatbath β=6.0, $16^3 \times 32$.
- MG benchmark: κ near 0.157–0.158 on configs A and B.

Backup: multigrid and operator applications

Slides 15–16 count **fine** (D) and **coarse** (D_c) applications during FGMRES ($\|r\| \le 10^{-10}$).

Algorithm: FGMRES with DD-αAMG V-cycle preconditioner

procedure V-CYCLE(x, b)
  $r \leftarrow b - Dx$ — fine
  for $i = 1, \ldots, n_{\text{pre}}$ **do**
    $x \leftarrow \text{Smooth}(x;\, D)$ — fine
  end for
  $r \leftarrow b - Dx$ — fine
  $r_c \leftarrow P^\dagger r$; solve $D_c e_c = r_c$ — coarse
  $x \leftarrow x + P e_c$
  for $i = 1, \ldots, n_{\text{post}}$ **do**
    $x \leftarrow \text{Smooth}(x;\, D)$ — fine
  end for
end procedure

Require: RHS b, initial x, prolongator P, coarse operator D_c
while $\|b - Dx\| > \text{tol}$ **do**
  V-CYCLE(x, b)
  FGMRES update using D — fine
end while

Benchmark: $n_{\text{pre}} = n_{\text{post}} = 4$; $D_c = P^\dagger D P$. Counts sum over all FGMRES iterations; setup is excluded.

Backup: DD-αAMG setup and solve parameters

`LatticeQCD.jl` DD-αAMG on $16^3 \times 32$; block size 4^4; $N_v \in \{12, 24, 48, 64\}$.

`setup_DDalphaAMG`

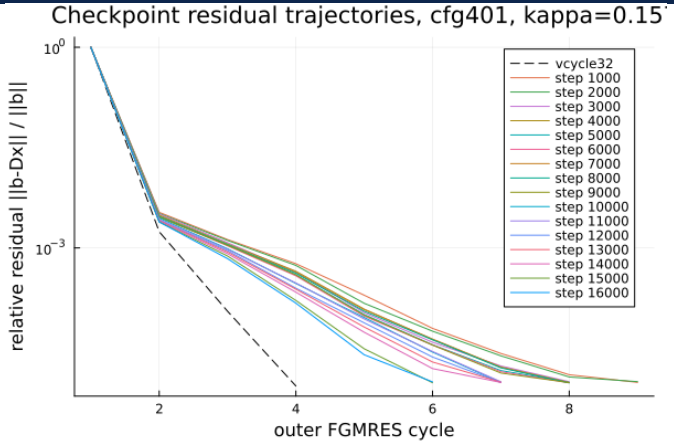
Parameter	Value
Shared	
Block size	4
`setup_smoother`	SAP (3 iters)
`setup_block_solver`	MR (4 steps)
`use_chirality`	`true`
Network vectors	
`setup_mode`	`:direct_near_null`
`coarse_vectors`	network output
V-cycle baseline	
`smoother`	`:vcycle`
`nvcycle`	6

`solve_DDalphaAMG` (FGMRES + V-cycle preconditioner)

Parameter	Value
`tol`	10^{-10}
`maxiter` / `restart`	200 / 40
`smoother`	SAP
`pre_smooth` / `post_smooth`	4 / 4
`sap_block_mr_steps`	4
`sap_schedule`	`:boundary8`
`coarse_tol`	10^{-1}
`coarse_maxiter` / `coarse_restart`	5 / 30
`coarse_odd_even` / `sap_odd_even`	`false`

Main-result slides count fine (D) and coarse (D_c) applications during `solve_DDalphaAMG` only; setup wall time is in the next backup slides.

Backup: checkpoint residual trajectories



Config A (cfg401), $\kappa = 0.157$, $K = 6$.
FGMRES relative residual vs. outer cycle at training checkpoints.
Dashed: classical adaptive setup with $N_v = 32$.

Preconditioning improves steadily; late checkpoints approach the classical baseline.

Backup: AMG setup wall time (config B)

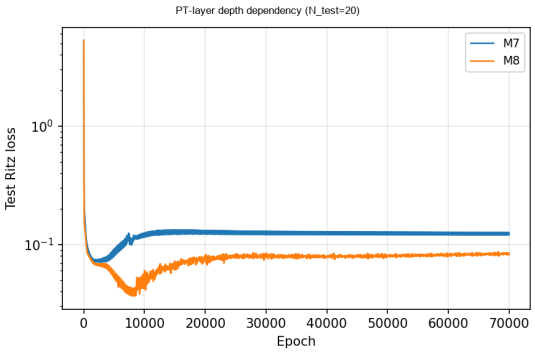
Config B (cfg450), $16^3 \times 32$.

Online wall time of `setup_DDalphaAMG`; offline network training excluded.

N_v	$\kappa = 0.157$		$\kappa = 0.1586$	
	Network	V-cycle	Network	V-cycle
12	41.1 s*	17.5 s	5.1 s	13.1 s
24	7.5 s	38.8 s	6.7 s	92.9 s
48	11.8 s	184.7 s	11.5 s	378.7 s
64	15.3 s	347.7 s	15.9 s	644.8 s

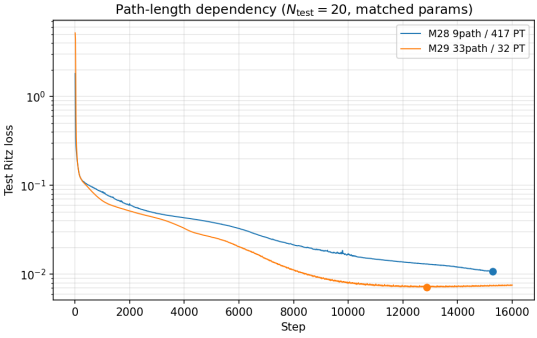
Network: checkpoint load, inference, chiral coarsening, coarse-operator build.
V-cycle: random vectors, 6 adaptive iterations, coarse build.
*First measured case; likely includes Julia/CUDA cold-start overhead.
Offline training to step 4000: $\sim$19–25 h per N_v (one-time, amortized).

Backup: architecture ablations



PT-layer depth dependency

$8^3 \times 16$



Path-length dependency

$16^3 \times 32$

- Deeper PT stacks lower the loss on configurations not used for training.
- Longer path sets help at matched parameter count (33 paths vs. 9 paths).