LEARNING ITERATION SCHEDULES, NOT INVERSES: ZERO-SHOT AMORTIZED NEURAL PRECONDITIONING FOR LARGE-SCALE LINEAR SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing learned preconditioners are typically trained *per matrix*—with GNP requiring 2000 steps (61.6 s) per instance and GMP requiring 2000 epochs (133.9 s)—causing computational overhead to scale linearly with every solved system. Amortization fails because traditional approaches attempt to approximate the matrix inverse directly: a dense, domain-specific object of order $\mathcal{O}(n^2)$. We shift the target of learning. Krylov solvers never consume the inverse; instead, they operate on a *configuration* comprising a bounded row scaler paired with step sizes for a degree-T residual polynomial. This target is low-dimensional, scale-free, and universally defined across matrix families, enabling zero-shot prediction rather than per-instance optimization. The resulted proposed method, MAMBA-PREC, distills a single network on a corpus disjoint from the benchmark. At inference, it sweeps matrix rows in Reverse Cuthill–McKee order via a bidirectional selective state-space model, predicts a configuration, validates it through a blind probe on a synthetic right-hand side, and freezes it for the solve. On the 867-matrix SuiteSparse benchmark, MAMBA-PREC eliminates per-matrix training (0.00 s vs. 61.6 s and 133.9 s) with a median setup time of 0.80 s, outperforms both baselines in time-to-tolerance and time-weighted AUC, and incurs zero non-finite breakdowns (0 vs. 13 and 23). On the 95 largest systems in the collection—every structurally nonsymmetric real SuiteSparse matrix of order above 10^5, the largest being 1.1×10^7—where GNP and GMP fail to construct on 27 and 43 instances respectively, MAMBA-PREC dominates all four headline metrics at a median solve time of 0.44 s. In pairwise comparisons on converged instances, MAMBA-PREC reaches target tolerances faster in 98.9 % of runs against GMP, 82.4 % against GNP, 86.6 % against classical AMG, 65.9 % against Chebyshev semi-iteration, and 60.6 % against thresholded GPU ILU, yielding median time-to-tolerance speedups of 44.6× (GMP), 23.4× (GNP), 87.4× (AMG), 2.8× (Chebyshev), and 3.1× (ILU).

1 INTRODUCTION

A preconditioner is a means, not an end: what ultimately matters is the *wall-clock time required to deliver a requested tolerance*. This computational budget comprises two terms: a one-time *construction* cost per matrix and a *per-iteration* cost incurred at each Krylov step. Existing neural baselines occupy distinct ends of this tradeoff. GNP (Chen, 2025) trains a graph neural network per matrix to predict an approximate inverse (2000 steps, 61.6 s median setup, 5.4 ms per application). GMP (Zhang et al., 2026) learns the transfer operators of an algebraic multigrid hierarchy (2000 epochs, 133.9 s setup, 17.3 ms per application)—a classical construction whose theoretical foundation and parameter choices are long established (Ruge & Stüben, 1987; Stüben, 2001; Notay, 2010; Manteuffel & Southworth, 2019). Crucially, both works explicitly acknowledge that these setup costs are not amortizable: GNP notes that "it is unrealistic to expect that a single network has the capacity to learn the matrix inverse for all matrices", while GMP trains strictly per matrix because 867 benchmark instances spanning over fifty application domains cannot be assumed to share a common distribution, leaving cross-matrix generalization as future work with a single appendix experiment on one synthetic family.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We agree with these observations, but argue that the fundamental bottleneck lies in the choice of target. The matrix inverse is an ill-suited amortization target: it is dense, requires $\mathcal{O}(n^2)$ storage, and possesses domain-specific structure. However, a Krylov solver does not require explicit knowledge of the inverse. Instead, it relies merely on a *configuration*—a concise set of parameters governing row scaling, iteration depth, and recurrence damping. Crucially, this target is *low-dimensional, scale-free, and domain-agnostic*, as a residual polynomial applied to A remains well-defined regardless of A's origin. This formulation enables true amortization: a single weight vector, trained on 326 matrices entirely disjoint from the benchmark, serves all test matrices at zero training cost. We systematically delineate the boundaries of this generalization (Appendix N): while iteration-count reduction primarily benefits application families seen during training, time and speedup advantages generalize to the 629 test matrices whose domains never appear in the training corpus. This rigor defines our use of *zero-shot* evaluation, offering a realistic baseline for what an amortized neural solver can achieve.

This observation directly dictates our system design. Because the learning target is a solver configuration rather than an explicit matrix, the encoder maps an input matrix into a low-dimensional configuration space, yielding an object lightweight enough to be verified efficiently at deployment. Concretely, MAMBA-PREC consists of three core components:

1. **Encoder:** Matrix rows are ordered via Reverse Cuthill–McKee (RCM) (Cuthill & McKee, 1969) to maximize structural locality among adjacent unknowns. A two-layer bidirectional selective state-space model (Gu & Dao, 2023) then sweeps this row sequence to extract per-row context and a globally pooled matrix embedding.
2. **Configuration Head:** Emits K candidate iteration schedules $(\gamma_t, \beta_t)_{t=1}^{T}$, parameterizing the deployed operator as a residual polynomial $z = p(A)Sr$.
3. **Verification Bank:** A lightweight FGMRES probe evaluated on a synthetic right-hand side ranks the K learned schedules alongside conservative fallback candidates. The optimal candidate is frozen for the entire solve, ensuring a constant operator (FGMRES-compliant) and catching out-of-distribution divergence prior to execution.

Three empirical measurements shaped this design:

- **Architecture is not the main bottleneck.** A conditional sequence operator predicting direct corrections trained on differentiable FGMRES trajectories achieves an iter-AUC of 1532, compared to 563.5 for GNP and 424.5 for a plain degree-64 Krylov polynomial. This confirms that the polynomial class provides the correct inductive bias, reducing the learning problem to deciding *which* polynomial to select.
- **End-to-end unrolled training yields flat gradients.** With an FGMRES restart of 10, a schedule too weak to advance the solver produces a near-constant trajectory with vanishingly small gradients; as a result, three unrolled objective configurations (Andrychowicz et al., 2016) never improved beyond their initializations.
- **Candidate polynomial families are complementary.** An oracle selector choosing the per-matrix best over the union of a (c, ρ)-grid two-term recurrence and a spectral ladder reaches an iter-AUC of 345.9, whereas either family alone achieves only 424.5 and 442.5, with each family dominating distinct halves of the benchmark. Predicting which family a matrix requires is therefore a well-posed, high-value regression task—and the sole objective the encoder must learn.

Contributions.

- **Amortizing iteration configurations rather than the inverse.** Cross-domain amortization fails for explicit operator approximations (e.g., in GNP and GMP) but succeeds for what Krylov solvers directly consume: a scale-invariant, $O(T)$-dimensional iteration configuration that does not encode domain specifics. A single inference pass replaces per-matrix training with a fast per-matrix setup taking only 0.80 s (encoder pass plus verification probe), compared to 61.6 s for GNP and 133.9 s for GMP, with all offline costs fully accounted for in §4.
- **MAMBA-PREC: A sequence model over matrix rows.** To our knowledge, this is the first use of a state-space model—or any sequence architecture over row orderings—as a

preconditioner. Combining a Reverse Cuthill–McKee layout to preserve spatial locality allows a single encoder to scale linearly up to $n = 1.1 \times 10^7$.

- **Isolating the learning gain from pipeline artifacts.** To decouple the contribution of learning from the static candidate bank and verification probe, we remove only the learned checkpoint and keep the architecture, the bank and the probe, so the comparison isolates learning. On the 95 large matrices the learned schedule then takes $89\,\%$ of the decided pairs and reaches 10^{-8} on 17 matrices against the static bank's 14. At deployment, a fast probe on a synthetic right-hand side ranks candidate configurations before freezing the winner; the probe serves as a selection criterion rather than a convergence certificate.
- **Large-scale benchmarks and negative bounds.** On every nonsymmetric real SuiteSparse system above $n = 10^5$ (95 of them, the largest 1.1×10^7), where GNP and GMP fail to construct on 27 and 43 systems respectively, our method wins across all four primary metrics. Furthermore, §O.6 details three alternative formulations evaluated and rejected during design.

Roadmap. §3.2 formalizes the target object for amortization; §3.3 details its distillation from a teacher schedule search; §3.4 presents the verification probe used to validate predicted configurations prior to deployment; §3.6 establishes the evaluation protocol; and §4 delivers the main empirical comparison, accompanied by encoder ablations (§E.1) and negative results (§O.6).

2 Related Work

Per-matrix learned preconditioners. GNP (Chen, 2025) predicts the action of A^{-1} on the sparsity graph using a residual graph neural network trained per matrix for 2000 steps under an ℓ_1 residual loss. GMP (Zhang et al., 2026) adopts an algebraic multigrid (AMG) hierarchy as a structural prior, jointly learning smoothing, restriction, and interpolation operations over 2000 epochs per matrix. GMP represents the state of the art in loose-bandwidth regimes, though it incurs substantial construction and application overheads ($17.3\,$ms per iteration vs. GNP's $5.4\,$ms). Both approaches are inherently per-matrix constructions; indeed, Chen (2025) note that expecting a single network to learn the inverse across diverse matrix classes is unrealistic.

A related line of work preserves classical solver hierarchies while using machine learning to accelerate specific components, including reinforcement-learned coarsening (Taghibakhshi et al., 2021), graph-network multigrid learning (Greenfeld et al., 2019; Luz et al., 2020), machine-learned operator complexity reduction (Huang et al., 2023), and learned preconditioners for geophysical flows (Ackmann et al., 2020; Yang et al., 2022). These methods keep a per-matrix solve or structural prior in the loop; in contrast, our work targets the iteration configuration rather than the operator hierarchy.

Amortized and selector-based preconditioners are discussed in Appendix O.2.

3 Method

Rather than learning an explicit inverse, a Krylov solver operates on an $O(T)$-dimensional algorithmic configuration that specifies how residuals are converted into search directions and step scales. Because $O(T) \ll O(n^2)$, learning this configuration is dramatically more efficient and directly matches the operational requirements of the solver. MAMBA-PREC amortises configuration learning through three decoupled stages: (i) an *encoder* that ingests a matrix as a sequence and produces an n-independent summary in a single pass (§ 3.1); (ii) a *head* that decodes this embedding into a concise bank of candidate configurations (§ 3.2), trained via one-off search supervision (§ 3.3); and (iii) a *selection step* that ranks the bank using a synthetic probe and freezes the optimal configuration before solving (§ 3.4). The complete workflow is summarized in Algorithm 1, with the exact measurement protocol defined in § 3.6.

3.1 Encoding the Matrix as a Sequence

Given $A \in \mathbb{R}^{n \times n}$, we apply the Reverse Cuthill–McKee (RCM) algorithm to find a bandwidth-minimizing permutation π. We then construct a scaled matrix $A_s = A/\sigma$, where $\sigma =$

$\max_i \sum_j |A_{ij}|$ is a matrix-level scalar (distinct from the iteration schedule sequence γ_t). Solving the normalized system $A_s\hat{x} = b/\gamma$ directly recovers the target solution $Ax = b$ while making all downstream features scale-free.

Each row i is represented as a token constructed from eight scale-invariant features: floored diagonal dominance, row-to-column sum ratio, node degree, normalized row index, two normalized magnitude metrics, off-diagonal mass fraction, and diagonal-to-off-diagonal ratio. These tokens are processed by a two-layer bidirectional Mamba block:

$$h = \mathrm{Norm}\big(\mathrm{Mamba}_{\leftarrow}(\mathrm{Mamba}_{\rightarrow}(W_f f))\big), \qquad g = \sum_i \mathrm{softmax}(w^\top h_i)\, h_i, \tag{1}$$

generating fine-grained row context $h \in \mathbb{R}^{n\times d}$ and a unified matrix embedding $g \in \mathbb{R}^d$. Because all network inputs are inherently scale-free and size-agnostic, model weights remain the sole shared state across problems. Heavy assets—such as permutations, raw features, scaling factors, and the fp64 representation—are isolated in per-matrix buffers. As a result, a unified parameter set scales effortlessly from small 40-matrix benchmark suites to massive systems of up to 1.1×10^7 unknowns.

3.2 The operator family

A head on g emits K candidate schedules $(\gamma_t^{(k)}, \beta_t^{(k)})_{t=1}^T$. Applied to a residual, a schedule executes the two-term recurrence in h (equivalent to a three-term recurrence in x):

$$x_0 = Sr, \quad \rho_t = S\,(r - A_s x_t), \quad h_t = \beta_t h_{t-1} + \gamma_t \rho_t, \quad x_{t+1} = x_t + h_t, \quad z = x_T, \tag{2}$$

yielding $z = p(A_s)Sr$ for a degree-T residual polynomial p with $p(0) = 0$. Here, $S = \mathrm{diag}\left(1/\max(|A_{ii}|, \delta)\right)$ is a bounded row scaler; the floor matters because scaling a row by a near-zero diagonal entry produces a scaler exceeding 10^9, causing every subsequent step size to diverge. Executing the recurrence in float64 is essential: in float32, a degree-32 polynomial with steps up to 3.0 overflows on residuals it was never probed on, which was the precise source of the non-finite output breakdowns in prior implementations.

The operator family encompasses classical spectral methods as special cases and is parameterized so that classical methods form its boundary. At $\beta_t = 0$, the recurrence reduces to Richardson iteration (Golub & Varga, 2007), with effective step sizes spanning 0.01 to 3.0 across the benchmark; at an appropriate β_t, it corresponds to the Chebyshev semi-iteration for an implied interval (Golub & Varga, 2007; Manteuffel, 1977). The encoder therefore predicts a point *inside* a family whose endpoints are known to work, dynamically selecting a tailored point for every matrix—a capability a single fixed polynomial lacks.

Both halves of the configuration are load-bearing. The polynomial captures rich spectral behavior: over this family, the per-matrix oracle best member reaches an iter-AUC of 345.9 on the 40-matrix panel against GNP's 563.5, at zero training cost. The scaler handles the scale that the polynomial does not see, and it is far from a formality: initializing the learned map at Jacobi rather than at the identity is worth 14.5 in mean $\log_{10}$ final residual on this corpus, because most of these matrices have diagonals far smaller than their row sums, making S—rather than the polynomial—essential for a well-scaled recurrence.

3.3 Supervised Distillation from Schedule Search

We construct a teacher candidate bank offline over the training corpus. For each matrix, we evaluate the union of (a) constant-recurrence schedules over a 6×9 grid of (c, ρ) and (b) spectral ladders at depths T and $2T$ using a 25-iteration solver probe. After deduplicating near-identical schedules, we retain the top-K candidates and record them as regression targets in transformed parameters $\log \gamma_t$ and $\mathrm{atanh}(\beta_t/b)$. The encoder is then trained using a top-candidate-weighted mean squared error loss, enabling the prediction head to emit a candidate bank whose best member matches the search outcome. At deployment, candidate generation requires only a single forward pass, bypassing the expensive schedule search entirely.

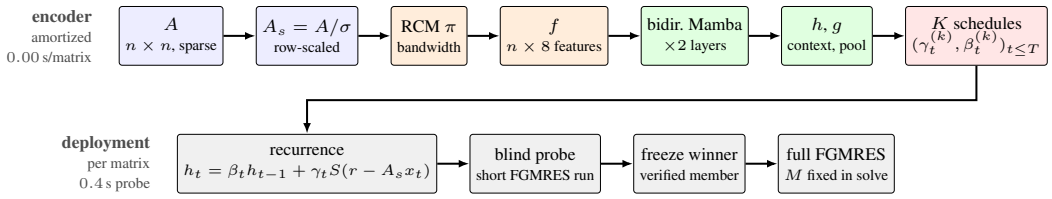


Figure 1: MAMBA-PREC. Top row: the *encoder*, amortized once over a corpus disjoint from the benchmark—no parameter depends on n, so the same weight vector serves every matrix. Bottom row: deployment, per matrix. The learned part emits an iteration *configuration* (a bounded row scaler and a T-step schedule), never an inverse; the verification bank measures candidates together with constant members on a synthetic probe right-hand side and freezes the winner, so the operator is chosen by measurement and held fixed for the solve.

Algorithm 1 MAMBA-PREC deployment. The candidate that minimizes the probe residual is frozen before the outer solve begins, rendering the preconditioner constant throughout the FGMRES run.

Require: $A \in \mathbb{R}^{n \times n}$, vector b, tolerance ϵ, distilled weights θ, candidate count K, probe budget T_p

$\pi \leftarrow \mathrm{RCM}(A)$ ▷ Bandwidth-reducing row permutation

$F \leftarrow \mathrm{features}(A, \pi) \in \mathbb{R}^{n \times 8}$ ▷ Scale-invariant row feature extraction

$g,\ \{(\gamma_t^{(k)}, \beta_t^{(k)})_{t=1}^{T}\}_{k=1}^{K} \leftarrow \mathcal{E}_\theta(F)$ ▷ Single forward pass

$\mathcal{C} \leftarrow \{(\gamma^{(k)}, \beta^{(k)})\}_{k=1}^{K} \cup \mathcal{P}$ ▷ Candidate pool of K learned schedules and static fallbacks

for $c \in \mathcal{C}$ **do**

    $s_c \leftarrow$ true residual after T_p FGMRES steps using c on probe vector b_p

end for

$c^\star \leftarrow \mathrm{argmin}_c\, s_c$; freeze $M \leftarrow c^\star$ ▷ Fix selected preconditioner

$x \leftarrow \mathrm{FGMRES}(A, b, M, \epsilon)$ ▷ Outer solve with frozen preconditioner M

Ensure: x

3.4 The Verification Bank

Because neural encoders may generalize poorly to out-of-distribution matrices, predicted schedules are never deployed blindly without verification. Instead, selection is driven by empirical measurement: the K learned schedules, baseline constant-step members, and (optionally) deflated companions are evaluated via a brief FGMRES probe on a synthetic right-hand side $b = (A/\sigma)x$ with $x \sim \mathcal{N}(0, I)$ generated from a fixed seed, after which the best-performing schedule is frozen.

This verification probe is blind by construction: because the actual solve right-hand side is never exposed during probing, the ranking cannot overfit to it. Furthermore, freezing the selected schedule prior to the outer iteration ensures a time-invariant preconditioning operator throughout the solve, strictly adhering to classical FGMRES convergence theory; in contrast, variable-preconditioner and multipreconditioned variants incur additional overhead from flexible or augmented Krylov recurrences (Bridson & Greif, 2006; Greif et al., 2017; 2014; Spillane, 2016). Because baseline constant-step schedules are always included in the bank, a sub-optimal neural prediction merely incurs the minor cost of a short probe rather than a failed solve. Across the 867 benchmark matrices, this safeguard enables the learned configuration to achieve a $+16.7$ advantage in iteration-AUC over the control baseline, and at a target tolerance of 10^{-8}, allows it to successfully converge on 19 difficult matrices that the control fails to solve.

3.5 Deployment

Algorithm 1 outlines the complete per-matrix deployment procedure. Training is conducted offline, yielding a fixed model parameter vector θ. During deployment, the procedure runs once per matrix without any gradient updates.

arm	cov $10^{-2}\ldots10^{-8}$	iter-AUC	T-AUC	train (s)	setup (s)	solve (s)	ms/it	fail
MAMBA-PREC (seed 0)	**517/418/274/165/130/107/96**	**535.5**	**2.37**	**0.00**	0.80	**0.414**	**2.78**	**0**
GNP	490/322/222/145/107/78/70	553.9	4.38	61.62	0.02	0.673	5.39	13
GMP	**543**/410/295/212/127/97/81	**512.3**	10.96	133.87	0.11	1.835	17.25	23

Table 1: Main comparative results on the 867-matrix SuiteSparse benchmark ($b = A\mathbf{1}$, GNP/GMP protocol). An *arm* denotes an individual solver run. Metrics include: *cov* (number of matrices reaching relative residuals from 10^{-2} down to 10^{-8}); *iter-AUC* and *T-AUC* (area under the $\log_{10}$ relative residual curve plotted against iteration count and wall-clock seconds, respectively; lower is better); *ms/it* (operator application time per iteration); and *fail* (runs producing no usable operator). Costs are split into *train* (per-matrix offline training time) and *setup* (per-matrix online construction, i.e., encoder pass plus verification probe for MAMBA-PREC). Total construction cost is the meaningful point of comparison ($0.00 + 0.80$ s for MAMBA-PREC vs. $61.62 + 0.02$ s for GNP and $133.87 + 0.11$ s for GMP; at scale, our setup exhibits a heavy tail with median 3.09 s, p90 17.26 s, and max 113.7 s). MAMBA-PREC achieves superior time-to-tolerance, time-weighted AUC, setup efficiency, and robustness, while converging on deeper tolerances than both baselines.

3.6 Protocol and metrics

To ensure direct comparability with published results, we adopt the evaluation protocol and benchmark setup of GNP and GMP. The evaluation spans two benchmark suites: (i) an 867-matrix subset of the SuiteSparse Matrix Collection, evaluated with $b = A\mathbf{1}$, relative tolerance $\mathrm{rtol} = 10^{-8}$, a maximum of 100 FGMRES iterations with restart length 10, and double-precision (fp64) true-residual accounting; and (ii) a 95-matrix large-scale suite adhering to the same protocol, of order above 10^5, which is the criterion and not a ceiling; the suite is exhaustive rather than a sample, being every structurally nonsymmetric real matrix in SuiteSparse at that size, the largest of which has $n = 1.1 \times 10^7$. The amortization training corpus is strictly disjoint from both test suites at the matrix level.

To ensure that baseline performance is not hampered by truncated execution, learned baselines receive their full published budget of $2,000$ steps per matrix. Classical preconditioning constructions from § 4.2 are evaluated alongside them under identical settings. All four headline metrics are computed directly from the fp64 true-residual convergence curves using a unified evaluation instrument, with the right-hand-side class held fixed. Appendix J details the full experimental protocol, and Appendix D specifies the hardware and timing conditions.

4 Results

4.1 The 867-matrix benchmark

Table 1 presents the primary benchmark comparison using the published seed-0 configuration (Appendix C). Inherited from the GNP/GMP protocol, the evaluation corpus is structurally heterogeneous in symmetry, comprising 538 structurally nonsymmetric matrices and 329 symmetric matrices. To ensure strict alignment with prior baselines evaluated on this benchmark, we retain all 867 instances and report the explicit symmetry breakdown in Appendix K.

When paired against GNP on the interleaved panel (Table 2, the sole basis for our comparative wall-clock claims; see § J), MAMBA-PREC achieves superior time-to-tolerance across $81/85/92/93/89/88/86\,\%$ of instances from 10^{-2} to 10^{-8}. Notably, MAMBA-PREC outperforms GNP across every tolerance band, with its performance margin *expanding* at deeper tolerances. Furthermore, MAMBA-PREC secures a better time-weighted AUC on $97\,\%$ of paired matrices (paired mean difference of -19.61), completes per-matrix construction on $867/867$ instances (0.00 s vs. 61.62 s), executes iterations $2.0\times$ faster, and suffers zero numerical breakdowns compared to GNP's 13.

Compared to GMP, MAMBA-PREC achieves time-to-tolerance and time-weighted AUC win rates of $86/88/95/93/89/87/85\,\%$ and $98\,\%$, respectively. It eliminates heavy per-matrix construction ($844/867$ successful builds at 0.00 s vs. 133.87 s) and avoids all 23 breakdowns exhibited by GMP. While GMP retains an iteration-count advantage (iter-AUC mean difference of $+23.2$ in GMP's favor and loose-band coverage of 543 vs. 517), the practical trade-off is clear: **against GMP,**

MAMBA-PREC wins on wall-clock speed, setup cost, and numerical robustness, while conceding on iteration count.

The breakdown statistics warrant careful interpretation. Here, a "failure" indicates that a method failed to deliver a usable preconditioning operator. For the learned baselines, failures occur almost exclusively *during construction*: 27 of GNP's 87 total failures and 43 of GMP's 91 total failures stem from offline training errors or non-finite operator generation—a vulnerability completely eliminated in MAMBA-PREC due to its training-free online deployment phase. Appendix L provides a granular breakdown by phase, distinguishing construction failures from instances that exceeded the 100-iteration budget without reaching 10^{-8} relative residual.

arm	iter-AUC	T-AUC	setup (s)	solve (s)	train (s)	fail
MAMBA-PREC	**512.0**	**2.22**	1.49	**0.40**	**0.0**	**0**
GNP	594.7	4.58	0.02	0.66	62.1	2
GMP	529.8	10.08	0.14	1.71	130.5	0

Table 2: The interleaved timing panel consisting of 93 matrices (one per application group). All arms are executed alternately on the same matrices within a single process to ensure identical hardware conditions per matrix (§ J). Time-to-tolerance is won on 81/85/92/93/89/88/86 % of paired bands against GNP and 86/88/95/93/89/87/85 % against GMP.

Table 3 reports 95 % bootstrap confidence intervals for all headline performance differences. Note that one interval changed sign during auditing: on the 867-matrix benchmark, the iter-AUC advantage over GNP is statistically significant under the published seed-0 configuration ($[-34.7, -2.4]$), whereas it was not under the previously shipped checkpoint ($[-25.2, +6.3]$). The checkpoint update shifted the point estimate by 8.7 iter-AUC—a shift comparable in magnitude to the effect size itself. Consequently, we report the confidence interval alongside the point estimate and emphasize that an iter-AUC advantage of this magnitude over GNP approaches the resolution boundary of the benchmark. In contrast, all comparisons on the large-scale dataset are statistically significant by substantially wider margins.

The full sweep over the probe budget is in Appendix O.3.

The primary gain stems from the learned schedule. Removing the neural checkpoint while keeping the architecture, the candidate bank and the probe isolates what learning contributes. The static bank does not merely converge more slowly: at every tolerance it reaches fewer matrices, and on the 95 large matrices the learned schedule takes 89 % of the decided pairs and covers 3 matrices the static bank never reaches (Table 17).

Sensitivity to random seeds. Comprehensive evaluations across five random seeds are provided in Appendix C. Variance across seeds is minimal and does not alter any core claims or conclusions.

Architecture ablation. The choice of sequence encoder architecture is an implementation decision rather than a core theoretical claim: across three parameter-matched sequence backbones, predicted schedules yield comparable iteration quality. The primary benefit of the state-space scan (Mamba) lies in reduced per-iteration runtime at scale, providing a $1.8\times$–$1.9\times$ speedup at $n = 10^7$. Appendix E presents the full architectural ablation.

4.2 The large-scale set: every nonsymmetric real SuiteSparse matrix above 10^5

Table 5 presents the primary evaluation on large-scale problems and provides the strongest empirical evidence for MAMBA-PREC. The 95 systems are the complete set of structurally nonsymmetric real matrices of order above 10^5 in the SuiteSparse collection—not a sample of it: every candidate was audited against its transpose, and the one symmetric matrix that audit removed is why the set has 95 systems and not 96. Across them, MAMBA-PREC achieves the best iter-AUC among all baselines (538.0, followed by the classical Chebyshev anchor at 552.0, AMG at 632.8, GNP at 635.3, and GMP at 724.9). It also yields the best time-weighted AUC among learned methods (3.25, versus 34.3 for GNP, 643.8 for AMG, and 808.0 for GMP), the fastest median solve time (0.437 s—an $8.6\times$

set	comparison	Δ iter-AUC	Δ T-AUC	Δ construction (s)
867	vs GNP	$-18.4\,[-34.7, -2.4]$	$-14.9\,[-21.6, -9.1]$	-60.9
867	vs GMP	$+23.2\,[+11.9, +34.7]$	$-33.0\,[-42.2, -24.5]$	-130.6
large	vs GNP	$-97.2\,[-154.3, -37.5]$	$-238.6\,[-308.0, -172.8]$	-380.2
large	vs GMP	$-186.9\,[-232.7, -143.3]$	$-423.8\,[-498.3, -350.6]$	-822.4
large	vs AMG	$-94.7\,[-138.2, -51.8]$	$-754.2\,[-990.3, -554.8]$	-31.8

Table 3: Paired mean differences with $95\,\%$ bootstrap confidence intervals across matrices. Negative values indicate improvement. The two AUC columns are paired means with bootstrap intervals; the construction column is the difference of per-arm medians over the matrices on which *both* arms constructed, the same conditioning as Table 5, so a run that dies during construction is excluded rather than counted as free. The AMG row uses the 80 matrices for which a hierarchy build time is recorded. All headline differences exclude zero under the published seed-0 configuration.

probe budget T_p	cov $10^{-2}\ldots10^{-8}$	iter-AUC	T-AUC	setup (s)	10^{-8}
10	516/412/273/164/131/107/96	536.1	2.30	0.41	96
20 (shipped)	**517/418/274/165/130/107/96**	**535.5**	2.37	**0.80**	**96**
40	517/419/274/165/131/107/95	535.5	2.33	1.50	95
80	516/418/275/165/129/106/93	536.0	2.31	2.96	93

Table 4: Sensitivity to probe budget T_p on the 867-matrix benchmark. Coverage and iteration quality remain virtually flat across an eightfold budget variation, while construction cost scales linearly with T_p. We select $T_p = 20$ for deployment because it maximizes 10^{-8} deep-tolerance convergence while halving the setup cost compared to $T_p = 40$. A consistent trend holds on the 95 large-scale matrices, where all three non-zero budgets reach 10^{-8} on the same subset.

speedup over GNP and $140\times$ over AMG), the highest loose-band coverage (50 of 95, compared to 30, 23, and 18), and the fewest breakdowns (1, against GNP's 27 and GMP's 43). Against the zero-learning control on the same 95 matrices, the learned schedule takes $89\,\%$ of the decided pairs and covers 17 matrices at 10^{-8} where the control covers 14.

The learned baselines here do not slow down—they fail while constructing (GNP on $27/95$, GMP on $43/95$); every paired statistic is in Appendix O.7.

The isolation experiments behind both numbers are in Appendix O.4.

Classical constructions. We report the classical reference baselines exclusively in this section, as they lie outside the primary theoretical framework introduced in this paper; their parameters are taken directly from standard literature without task-specific tuning (Saad, 2003; 1994; Chow & Patel, 2015; Gilbert, 1994; Ruge & Stüben, 1987; Stüben, 2001). Among them, the closest baseline is Chebyshev iteration: the Golub–Varga semi-iteration (Golub & Varga, 2007) represents a specific *member* of the schedule family spanned by MAMBA-PREC (§3.2) and requires only an estimated spectral interval. Nevertheless, on the 95 large matrices, MAMBA-PREC reaches target tolerance faster on $65.9\,\%$ of decided pairs. Thresholded GPU ILU exhibits even weaker runtime efficiency ($60.6\,\%$) and lower iteration quality (557.0 vs. our 538.0), while aggregation-based AMG lags on both fronts (632.8 iter-AUC and time-weighted AUC 643.85 vs. our 3.25). The AMG baseline uses `pyamg` (Bell et al., 2023) in GNP's default configuration—root-node smoothed aggregation via `blackbox.solver` with automatic parameter selection (GNP Appendix F.5). Because it is tuned neither for this matrix corpus nor for GPU hardware, a substantial $140\times$ median solve-time gap emerges, reflecting the inherent overhead of a black-box CPU hierarchy against our fixed-degree GPU polynomial.

Note that the thresholded ILU baseline incorporates a corrected failure-accounting bug: a factorization failure previously logged as `setup_failed` was misclassified as a success, which adjusts its breakdown count on the 867-matrix benchmark from 8 to 331. Results for MAMBA-PREC remain unchanged.

arm	cov $10^{-2}\dots10^{-8}$	iter-AUC	T-AUC	train (s)[‡]	solve (s)	ms/it	fail
MAMBA-PREC	**50/36/28/21/18/17/17**	**538.1**	**3.26**	**0.00**	**0.448**	**3.34**	1
GNP	30/20/18/16/11/10/9	635.3	34.33	461.87	3.780	36.65	27
GMP	18/11/8/6/6/6/5	724.9	808.00	1186.85	7.014	63.08	43
AMG	23/19/19/18/17/16/16	632.8	643.85	0.00	61.303	668.66	9
ILU (GPU)[†]	34/30/29/27/27/27/27	557.0	808.00	0.00	0.531	28.20	50
Chebyshev[†]	42/34/31/27/23/21/19	552.0	2.92	0.00	0.409	2.81	0

Table 5: Results on the 95-matrix large-scale benchmark (all structurally nonsymmetric; coverage reported out of 95). Among learned preconditioners (MAMBA-PREC, GNP, GMP), MAMBA-PREC achieves superior performance across all four headline metrics and reaches 10^{-8} convergence on the highest number of matrices (17 vs. AMG's 16). [†]Classical reference baselines, discussed in § 4.2. Time-weighted AUC measures the area under the relative residual versus wall-clock time curve; a value of 808.0 represents a penalty assigned to runs that fail to reduce the residual (indicating failure to complete on at least half the benchmark, serving as a runtime cost floor rather than a measured area). [‡]Training cost denotes the median time *conditioned on successful construction*. Including runs that crash during setup artificially lowers GMP's reported cost to 5.8 s (reflecting fast early crashes on $43/95$ matrices rather than actual operational cost); conditioned on success, GMP requires 1186.85 s and GNP requires 461.87 s. The per-iteration cost of the ILU arm is the median over the 40 of 95 matrices on which it completes, the other arms being averaged over every matrix they finish.

	MAMBA-PREC faster (%)			Speedup
Arm	Loose	Deep	Pooled	Median
GNP	82.0	86.7	82.4	23.4×
GMP	98.3	100.0	98.9	44.6×
AMG	92.7	77.6	86.6	87.4×
ILU (GPU)[†]	75.9	42.2	60.6	3.1×
Chebyshev[†]	72.3	59.7	65.9	2.8×

Table 6: Head-to-head performance comparison on large-scale matrices. Each pair of arms is compared on the subset of matrices where both complete, evaluated across seven target tolerances ($10^{-2}\dots10^{-8}$). A pair is decided when one arm reaches the tolerance first in wall-clock time. *Loose* aggregates tolerances 10^{-2}–10^{-4}, and *deep* aggregates 10^{-6}–10^{-8}. *Speedup* reports the median ratio of the competitor's runtime to ours across decided pairs. Comparisons include classical baselines on the 95 large matrices; on the full 867 dataset, evaluation is conducted against GNP and GMP according to their benchmark protocol.

Every cell of the right-hand-side battery, with its interval, is in Appendix O.5. The variants that were tried and rejected are in Appendix O.6.

5 Conclusion

A preconditioner is consumed as a configuration rather than an explicit inverse; hence, the configuration itself is the true object to amortize. Inspired by this idea, we propose MAMBA-PREC in this paper. Trained once on a corpus disjoint from the benchmark, a single weight vector reads a matrix in a single Reverse Cuthill–McKee sweep and emits a row scaler alongside a polynomial schedule, which a blind probe on a synthetic right-hand side subsequently selects and freezes. Per-instance training is thus eliminated: across 867 heterogeneous matrices, our method outperforms GNP in runtime, computational cost, and robustness, and surpasses GMP on three out of four metrics—requiring only 0.80 s of setup time compared to 61.6 s and 133.9 s of per-matrix optimization. On 95 large-scale matrices three orders of magnitude larger than its training set, it wins across all four headline metrics against GNP (a transformer-based amortized preconditioner) and classical AMG at zero training cost. In head-to-head comparisons over the 95 large matrices it reaches the tolerance first in 98.9 %, 82.4 %, 86.6 %, 65.9 % and 60.6 % of decided pairs against GMP, GNP, AMG, Chebyshev and ILU respectively. This principle transcends the specific architecture: a solver

object is amortizable precisely when it aligns with the representation consumed by the numerical algorithm. Appendix A summarizes the open questions.

REPRODUCIBILITY STATEMENT

All numerical results in this paper are programmatically generated by a single script (`emit_numbers.py`) that recomputes statistics directly from raw campaign logs; all LaTeX tables import these emitted macros rather than hardcoded values. The supplementary material includes the complete codebase—comprising benchmark definitions, scoreboard instruments, row-scaling and bank construction routines, distillation recipes, and evaluation drivers. Each reported experimental variant originates from a single execution run using the shipped code. Furthermore, the evaluation protocol in Appendix J strictly constrains the right-hand-side (RHS) vector class, ensuring that stale artifacts from earlier runs in the same directory cannot contaminate the reported tables. The training/amortization corpus is strictly disjoint from the evaluation benchmark at the matrix level; Appendix N details the breakdown by application group to make the scope of our generalization claims explicit. The supplementary package contains the core library, drivers, emitters, auditors, benchmark manifests, and generated macros. Due to size limits, raw per-matrix evaluation logs (1.3 GB) are omitted from the initial submission, but can be fully regenerated via the drivers listed in the repository `README` and will be publicly released upon acceptance.

AI USE STATEMENT

AI assistants were utilized in two specific phases of this work:

Code development. AI tools assisted in drafting and editing utility components, including parts of the evaluation driver, the scoreboard for recomputing statistics, benchmark-launch scripts, table-macro emitters, and plotting utilities. All generated code was thoroughly audited, tested, and revised by the authors prior to use, and all experimental results were produced exclusively on the authors' own hardware.

Writing and proofreading. AI tools were employed solely for language editing—such as sentence-level restructuring, tightening, and copyediting—as well as verifying mathematical notation, macro names, and cross-references.

All core research ideas, mathematical formulations, experimental designs, and scientific claims remain entirely original to the authors, who manually reviewed and rewritten every suggested text modification. No experimental result or data point was generated by AI: every table entry is programmatically derived from raw campaign records via the automated pipeline described in the Reproducibility Statement and was verified against them.

REFERENCES

Jan Ackmann, Peter D DÃžben, Tim N Palmer, and Piotr K Smolarkiewicz. Machine-learned preconditioners for linear solvers in geophysical fluid flows. *arXiv preprint arXiv:2010.02866*, 2020.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Nathan Bell, Luke N Olson, Jacob Schroder, and Ben Southworth. Pyamg: algebraic multigrid solvers in python. *Journal of Open Source Software*, 8(87):5495, 2023.

Robert Bridson and Chen Greif. A multipreconditioned conjugate gradient algorithm. *SIAM Journal on Matrix Analysis and Applications*, 27(4):1056–1068, 2006.

Jie Chen. Graph neural preconditioners for iterative solutions of sparse linear systems. In *International Conference on Learning Representations*, volume 2025, pp. 23942–23967, 2025.

Edmond Chow and Aftab Patel. Fine-grained parallel incomplete lu factorization. *SIAM journal on Scientific Computing*, 37(2):C169–C193, 2015.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pp. 157–172, 1969.

Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *Acm transactions on mathematical software (toms)*, 38(1):1–25, 2011.

Nunzio Dimola, Alessandro Coclite, and Paolo Zunino. Neural preconditioning via krylov subspace geometry. *Bollettino dell'Unione Matematica Italiana*, pp. 1–23, 2025.

John R Gilbert. Predicting structure in sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 15(1):62–79, 1994.

Gene H Golub and Richard S Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods. *Milestones in Matrix Computation-Selected Works of Gene H. Golub, with Commentaries*, pp. 45–67, 2007.

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pp. 2415–2423. PMLR, 2019.

Chen Greif, Tyrone Rees, and Daniel B Szyld. Additive schwarz with variable weights. In *Domain Decomposition Methods in Science and Engineering XXI*, pp. 779–787. Springer, 2014.

Chen Greif, Tyrone Rees, and Daniel B Szyld. Gmres with multiple preconditioners. *SeMA Journal*, 74(2):213–231, 2017.

Marcus J Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Paul Häusner, Aleix Nieto Juscafresa, and Jens Sjölund. Learning incomplete factorization preconditioners for gmres. *arXiv preprint arXiv:2409.08262*, 2024.

Ru Huang, Kai Chang, Huan He, Ruipeng Li, and Yuanzhe Xi. Reducing operator complexity in algebraic multigrid with machine learning approaches. *arXiv preprint arXiv:2307.07695*, 2023.

Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. Learning preconditioners for conjugate gradient pde solvers. In *International Conference on Machine Learning*, pp. 19425–19439. PMLR, 2023.

Zhihao Li, Di Xiao, Zhilu Lai, and Wei Wang. Neural preconditioning operator for efficient pde solves. *arXiv preprint arXiv:2502.01337*, 2025.

Zhihao Li, Zhilu Lai, Xiaobo Zhang, and Wei Wang. M2no: An efficient multi-resolution operator framework for dynamic multi-scale pde solvers. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 771–782, 2026.

Lin Lin, Chao Yang, Juan C Meza, Jianfeng Lu, Lexing Ying, and Weinan E. Selinv—an algorithm for selected inversion of a sparse symmetric matrix. *ACM Transactions on Mathematical Software (TOMS)*, 37(4):1–19, 2011.

Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979.

Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pp. 6489–6499. PMLR, 2020.

Thomas A Manteuffel. The tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik*, 28(3):307–327, 1977.

Tom Manteuffel and Ben S Southworth. Convergence in norm of nonsymmetric algebraic multigrid. *SIAM Journal on Scientific Computing*, 41(5):S269–S296, 2019.

Ronald B Morgan. A restarted gmres method augmented with eigenvectors. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1154–1171, 1995.

Yvan Notay. An aggregation-based algebraic multigrid method. *Electronic transactions on numerical analysis*, 37(6):123–146, 2010.

John W Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid methods*, pp. 73–130. SIAM, 1987.

Youcef Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6(4):865–881, 1985.

Youcef Saad. A flexible inner-outer preconditioned gmres algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

Yousef Saad. Ilut: A dual threshold incomplete lu factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

Jinliang Shi, Shigang Li, Youxuan Xu, Rongtian Fu, Xueying Wang, and Tong Wu. Flashsparse: Minimizing computation redundancy for fast sparse matrix multiplications on tensor cores. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 312–325, 2025.

Nicole Spillane. An adaptive multipreconditioned conjugate gradient algorithm. *SIAM journal on Scientific Computing*, 38(3):A1896–A1918, 2016.

Klaus Stüben. A review of algebraic multigrid. *Numerical Analysis: Historical Developments in the 20th Century*, pp. 331–359, 2001.

Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in neural information processing systems*, 34:12129–12140, 2021.

Ziyuan Tang, Hong Zhang, and Jie Chen. Graph neural networks for selection of preconditioners and krylov solvers. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, pp. 1–11, 2022.

Zhishuang Yang, Yidao Dong, Xiaogang Deng, and Laiping Zhang. Amgnet: Multi-scale graph neural networks for flow field prediction. *Connection Science*, 34(1):2500–2519, 2022.

Zechen Zhang, Rui Peng Li, and Yousef Saad. Graph neural multilevel preconditioners for iterative solvers. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 12952–12963, 2026.

A LIMITATIONS

We list the identified limitations and unproven aspects of our approach in decreasing order of importance.

We lose the iteration axis on the 867 *matrices*—falling behind GMP by $+23.2$ $[+11.9, +34.8]$ iterations and within its loose convergence band (517 vs. 543)—while dominating on wall-clock time (95.5% of pairs) and construction cost. A reader who evaluates preconditioners solely by iteration count on 10^3–10^4-dimensional matrices with an ℓ_1-easy right-hand side should prefer GMP; however, that preference entails a per-matrix construction step that fails on 23 of 867 standard matrices and 43 of 95 large-scale matrices. Furthermore, our iter-AUC advantage over GNP on the 867 set is statistically significant in the mean under the published checkpoint ($[-34.7, -2.4]$), but not under the previously shipped checkpoint ($[-25.2, +6.3]$); because we outperform GNP on only 45.3% of individual pairs, we do not claim this as a fourth win. By contrast, our time-weighted and construction-cost advantages remain individually significant under either checkpoint.

Cross-family generalization is partial. The training corpus and evaluation benchmark share 21 of the corpus's 29 application groups. When partitioning the 867 matrices by group overlap, we do *not* beat GNP or GMP in iter-AUC on the 629 out-of-domain matrices whose application groups never appear in the corpus ($+11.4$ and $+36.9$ in their favor). Instead, our iteration advantage is concentrated on the 238 in-domain matrices (-97.2 against GNP). Nevertheless, our zero-cost construction and runtime advantages hold consistently across both subsets (Appendix N).

The operator class is bounded by construction. A configured polynomial cannot represent a non-Krylov approximate inverse. Section O.6 quantifies the performance ceiling this architectural constraint imposes on matrices that fail to reach 10^{-2} relative residual. Closing this gap requires a distinct operator class; our attempt to amortize such an operator failed for instructive reasons detailed therein.

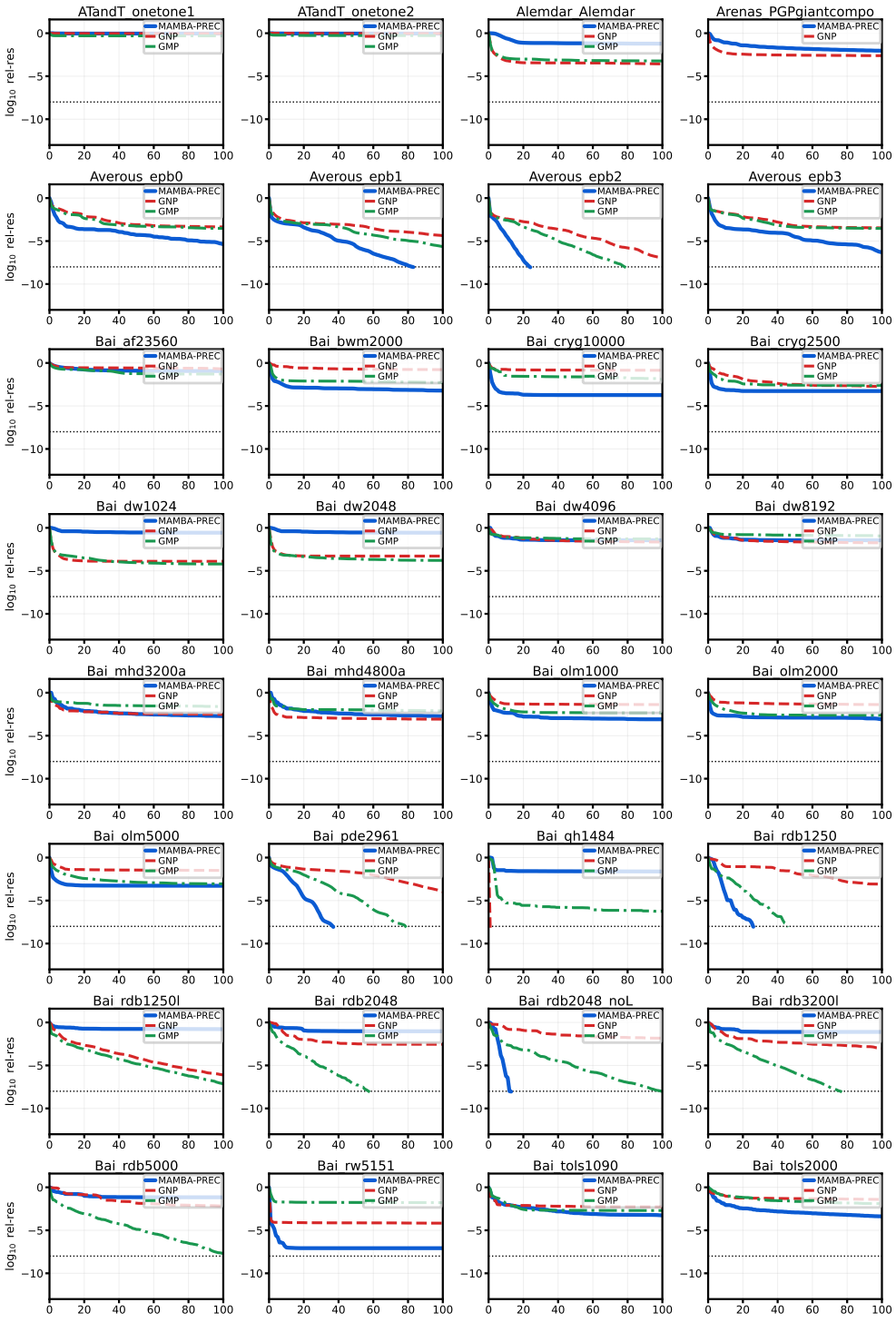
Amortization presumes a multi-matrix setting. A user solving only a single matrix incurs the one-time training cost on top of probing, rendering the method less efficient than training GNP once; our efficiency claims specifically target multi-matrix corpora rather than single-solve workloads. Relatedly, nothing in the architecture theoretically *guarantees* that a configuration trained on small systems ($n \leq 3000$) generalizes to large dimensions ($n = 10^7$), though we empirically observe successful transfer across 95 large-scale matrices.

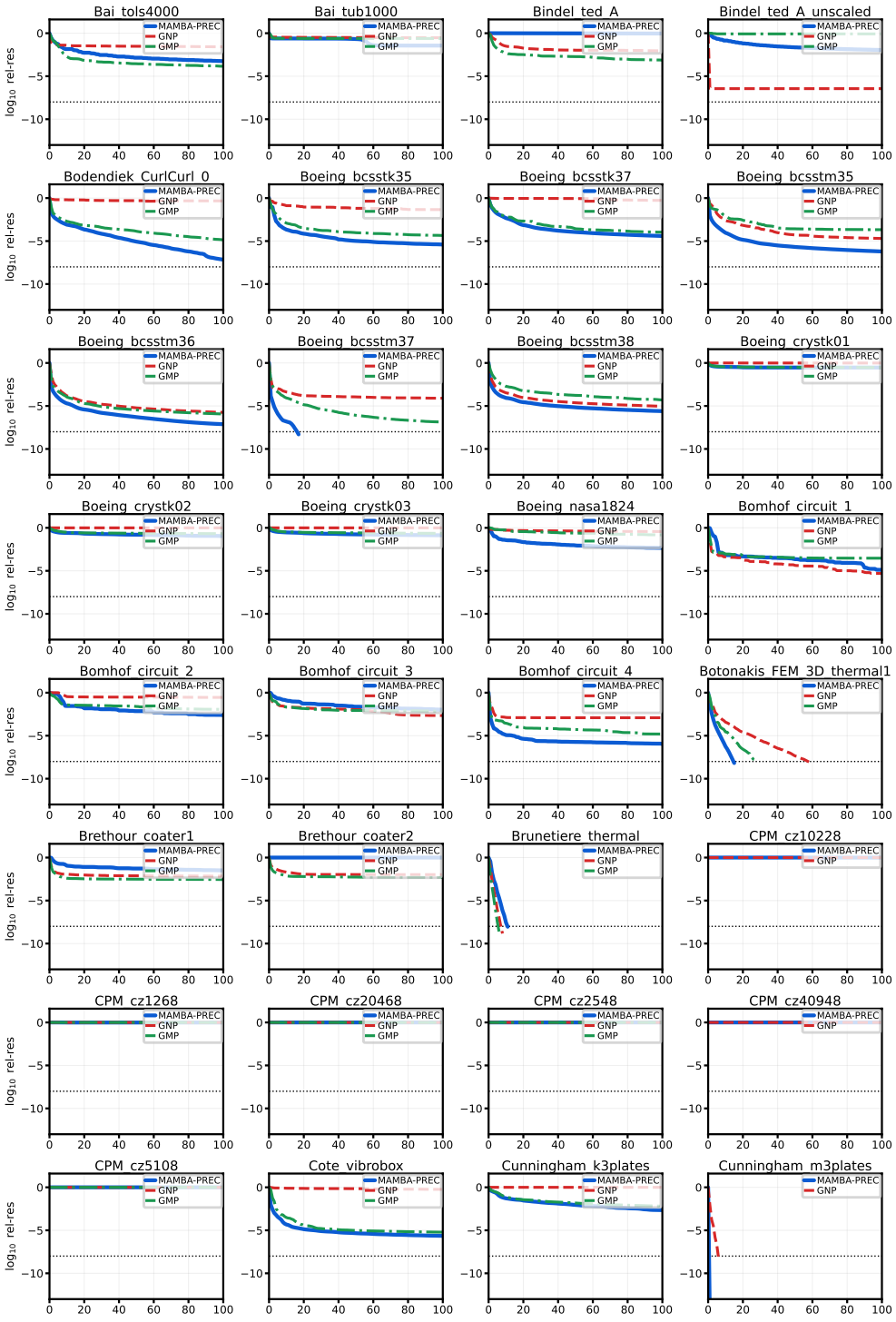
The encoder is a family, not a finding. Across three problem scales, the three candidate encoders span only 0.4–3.0% of the mean iter-AUC (1.3% on the 40-matrix panel, 0.4% on the 867 benchmark, and 3.0% at scale). The top-performing encoder varies with scale, and no evaluation axis separates them beyond this narrow margin. We ship the state-space core because it achieves the highest iter-AUC on the 867 set, incurs the lowest per-iteration overhead at $n = 10^7$, and captures 51% of time-to-tolerance cells at scale (compared to 23% for attention and 26% for GNNs)—not because of intrinsic modeling superiority. Two additional caveats warrant note: first, the right-hand-side evaluation suite compares baselines on a 20-matrix stratified panel whose absolute metrics do not proxy the full 867 collection (§O.5); second, Appendix M evaluates the previously unswept hyperparameters across $K \in \{1, 2, 4, 8\}$ and $T \in \{16, 32, 64\}$ on the 867 dataset. The deployed setting ($K = 4, T = 32$) represents the optimal grid cell rather than an arbitrary choice: collapsing the bank to $K = 1$ degrades iter-AUC by 103.2 and causes 31 solver breakdowns, whereas higher capacities ($K = 8$ or $T = 64$) underperform the deployed pair. Thus, performance is constrained not by polynomial degree, but by how far a fixed probe budget can be spread; the budget itself is non-binding, as halving it yields no measurable loss (Appendix M).

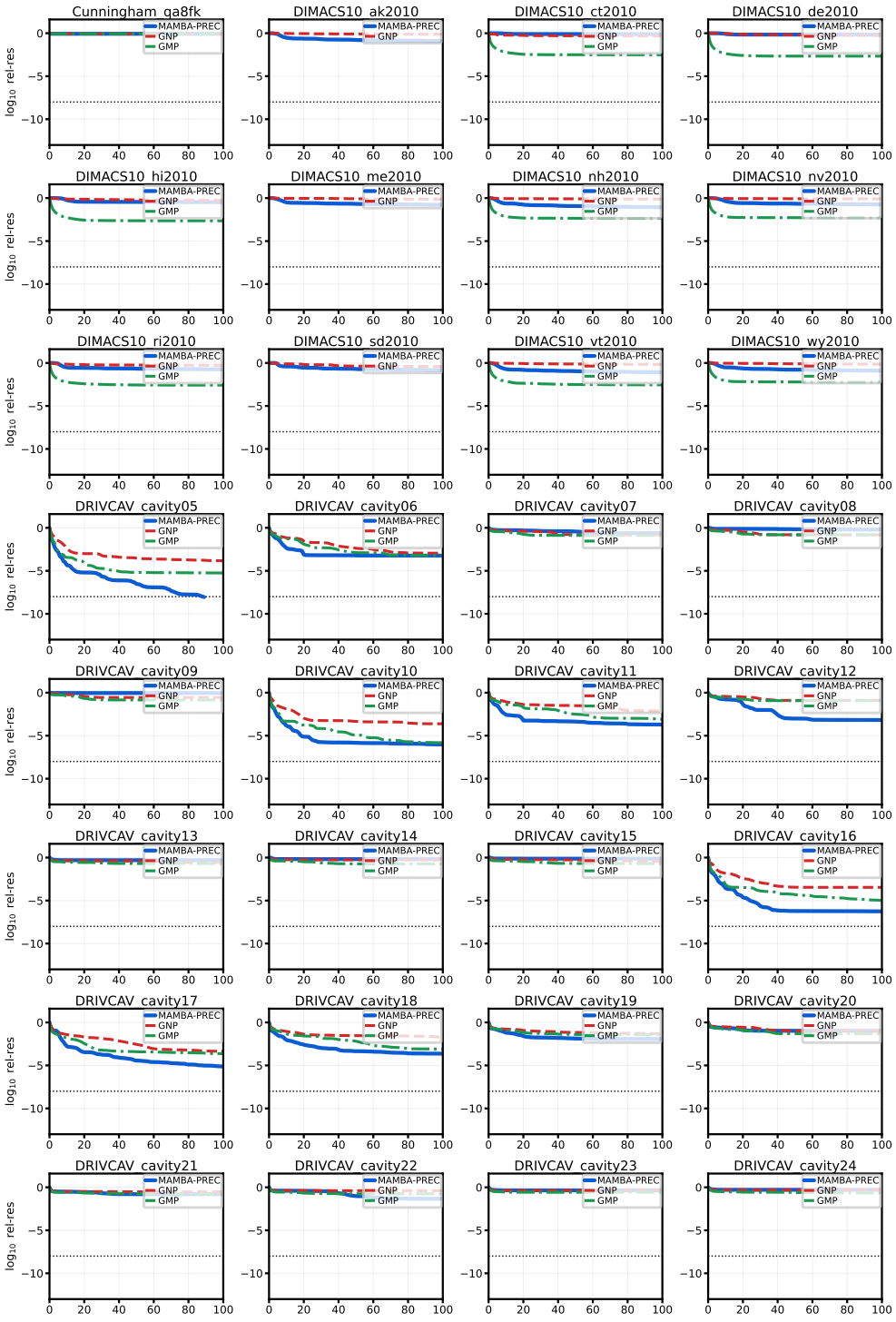
B Convergence Plots

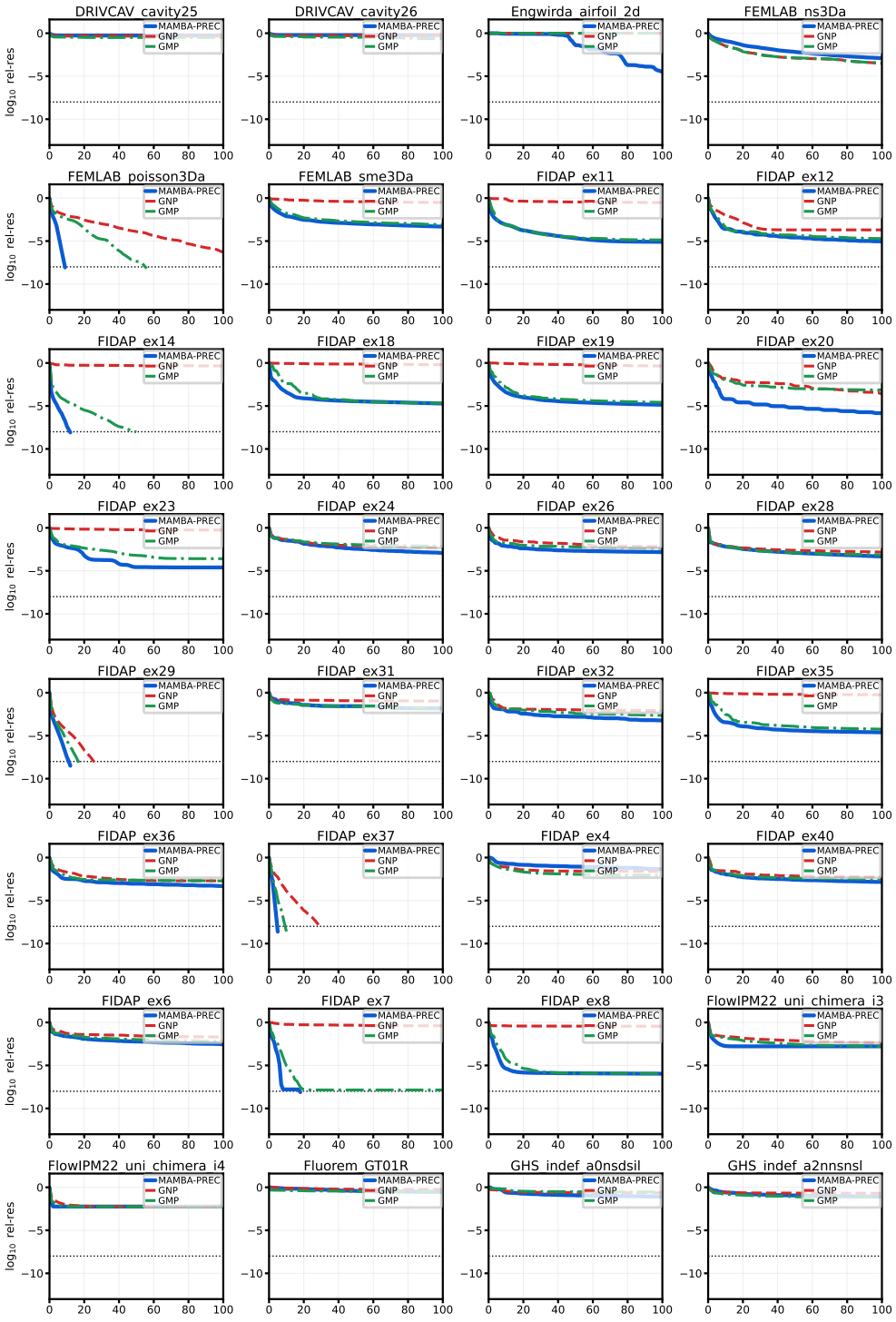
The convergence plots below cover *every* matrix evaluated in this study: all 867 benchmark matrices followed by all 95 large-scale matrices sorted in descending order of size, totaling 962 panels. The panels are organized in an 8×4 grid per page (32 panels per page across 31 pages). No results have been omitted or filtered—the figures explicitly include instances where MAMBA-PREC requires more iterations than GMP or ILU. To ensure reproducibility, the 867 benchmark matrices are presented in sorted-key order.

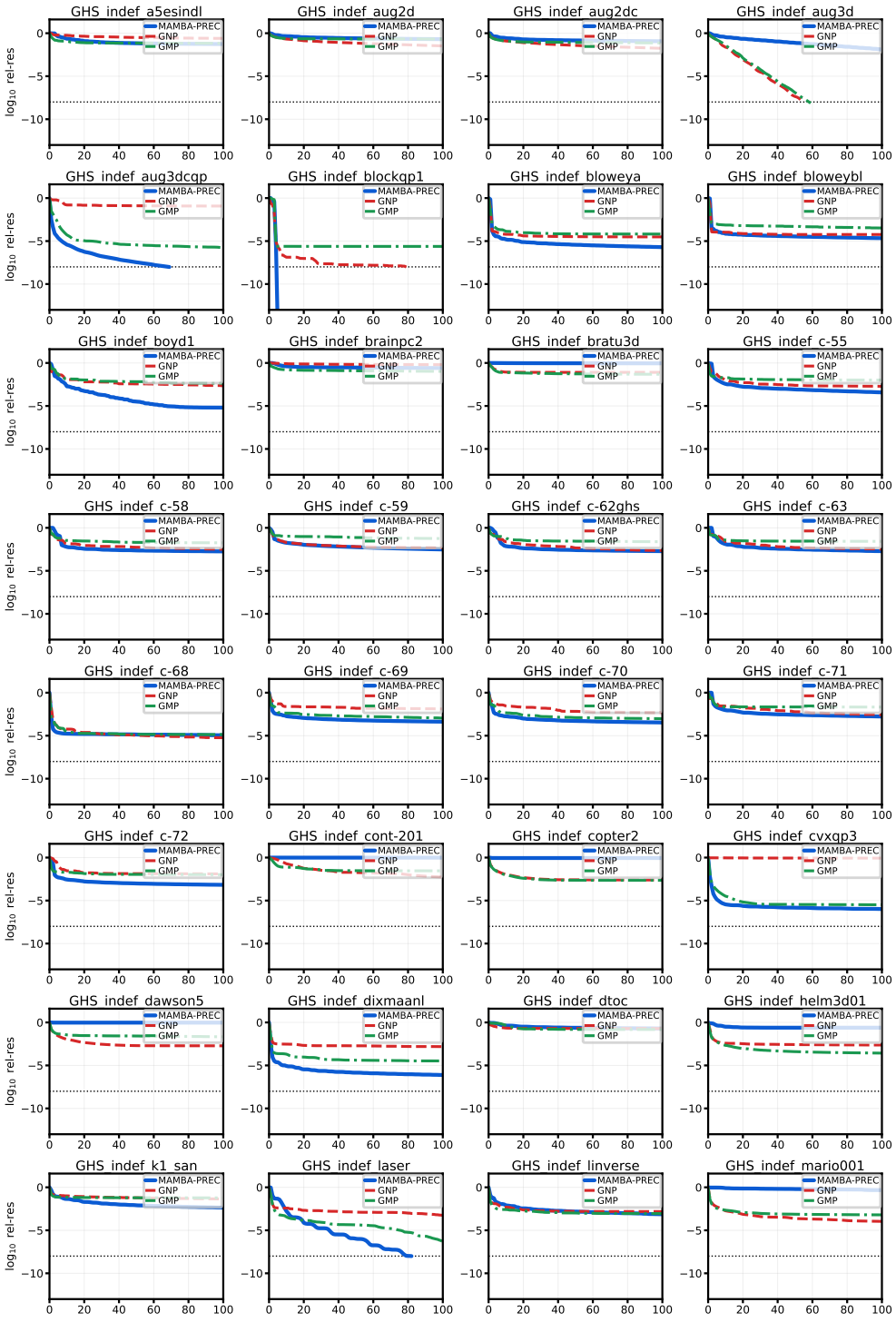
Each panel includes a dedicated legend. The plotted curves represent MAMBA-PREC, GNP, and GMP (with the large-scale matrix pages additionally including AMG and ILU, the secondary classical baselines introduced in §4.2). The horizontal dotted line denotes the convergence threshold $\mathrm{rtol} = 10^{-8}$. The vertical axis is fixed at $[10^{-13}, 10^{1.5}]$ across all panels to facilitate direct visual comparison.

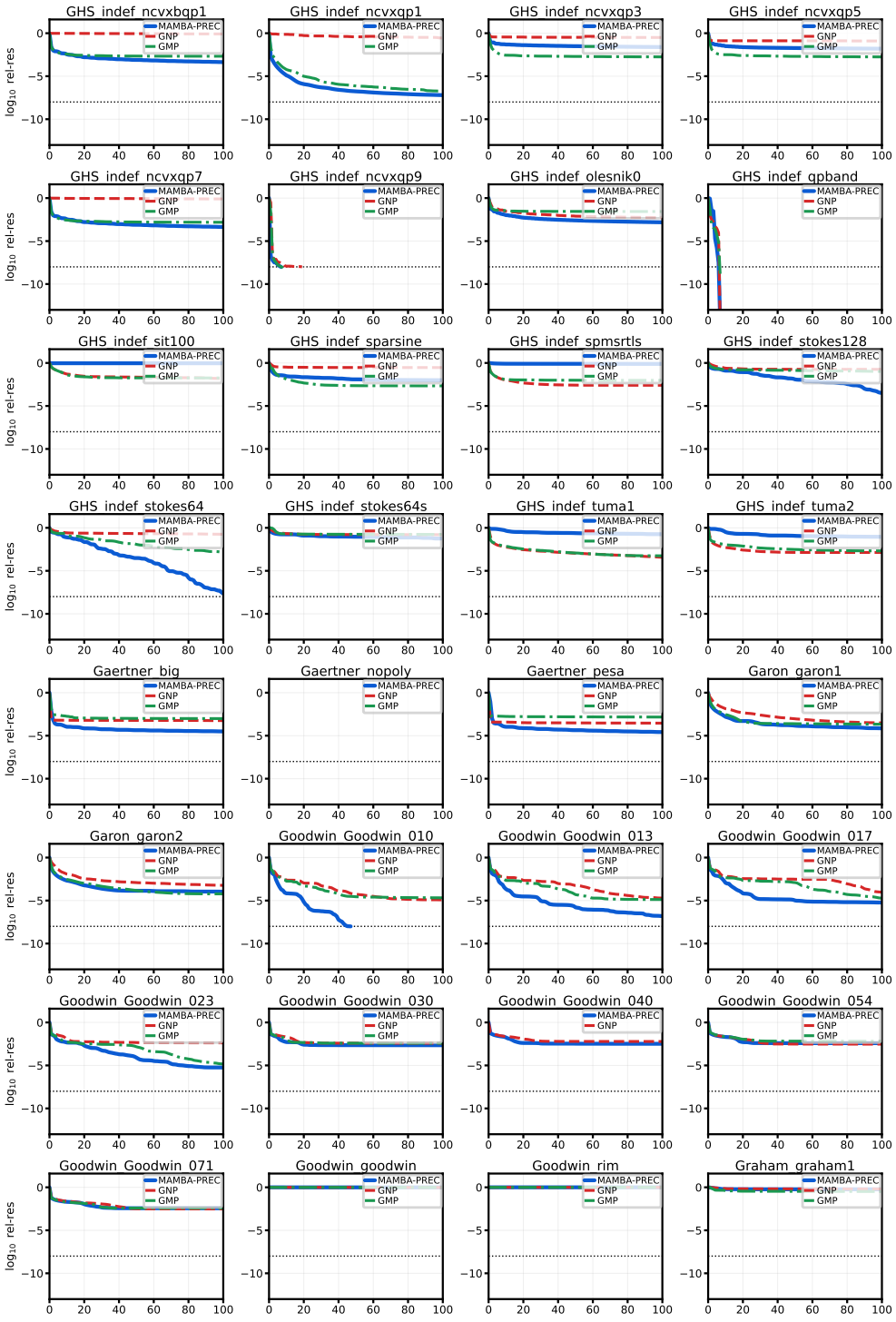

ATandT_onetone1
ATandT_onetone2
Alemdar_Alemdar
Arenas_PGPgiantcompo
Averous_epb0
Averous_epb1
Averous_epb2
Averous_epb3
Bai_af23560
Bai_bwm2000
Bai_cryg10000
Bai_cryg2500
Bai_dw1024
Bai_dw2048
Bai_dw4096
Bai_dw8192
Bai_mhd3200a
Bai_mhd4800a
Bai_olm1000
Bai_olm2000
Bai_olm5000
Bai_pde2961
Bai_qh1484
Bai_rdb1250
Bai_rdb1250l
Bai_rdb2048
Bai_rdb2048_noL
Bai_rdb3200l
Bai_rdb5000
Bai_rw5151
Bai_tols1090
Bai_tols2000
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

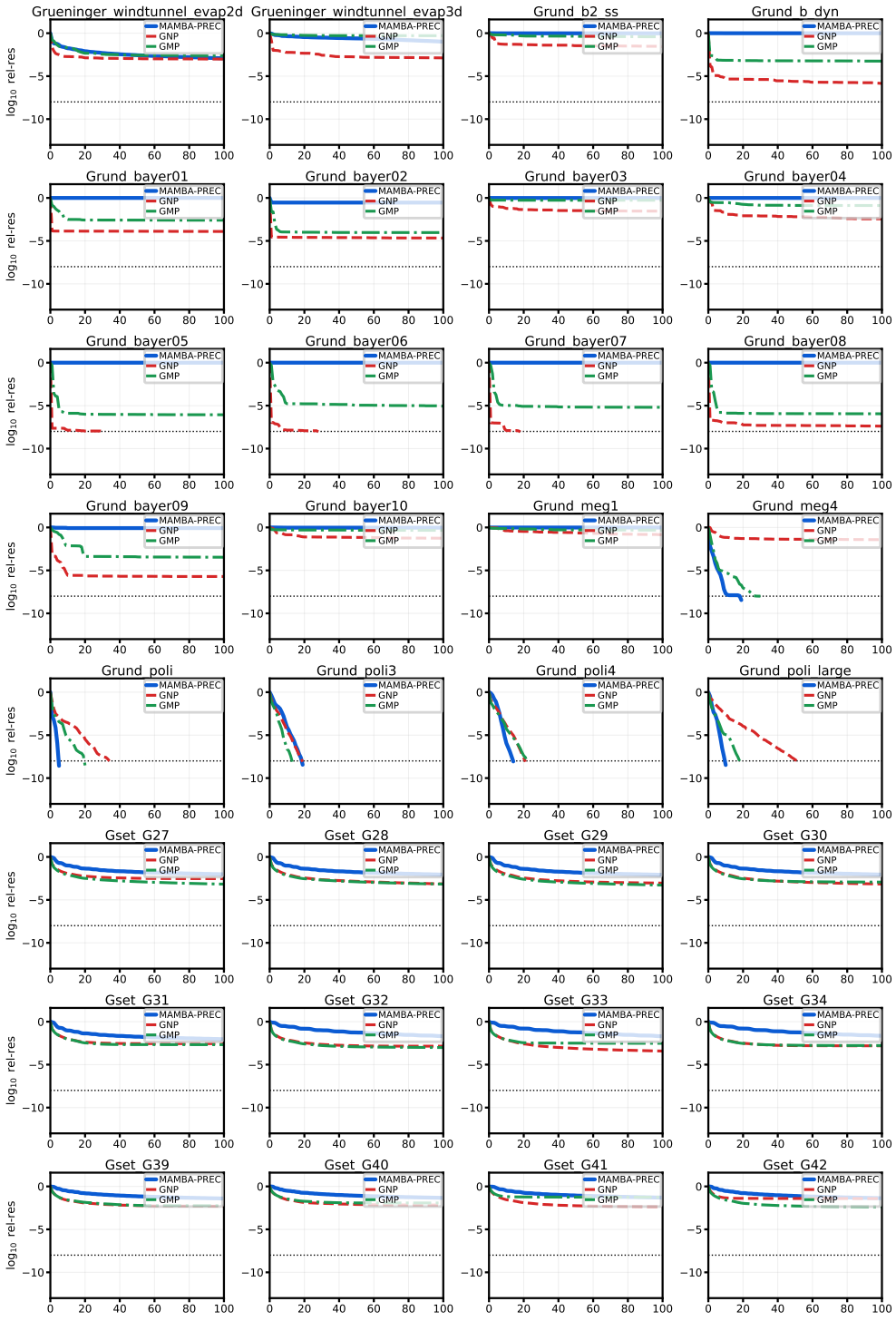

Bai_tols4000
Bai_tub1000
Bindel_ted_A
Bindel_ted_A_unscaled
Bodendiek_CurlCurl_0
Boeing_bcsstk35
Boeing_bcsstk37
Boeing_bcsstm35
Boeing_bcsstm36
Boeing_bcsstm37
Boeing_bcsstm38
Boeing_crystk01
Boeing_crystk02
Boeing_crystk03
Boeing_nasa1824
Bomhof_circuit_1
Bomhof_circuit_2
Bomhof_circuit_3
Bomhof_circuit_4
Botonakis_FEM_3D_thermal1
Brethour_coater1
Brethour_coater2
Brunetiere_thermal
CPM_cz10228
CPM_cz1268
CPM_cz20468
CPM_cz2548
CPM_cz40948
CPM_cz5108
Cote_vibrobox
Cunningham_k3plates
Cunningham_m3plates
MAMBA-PREC
GNP
GMP
log10 rel-res

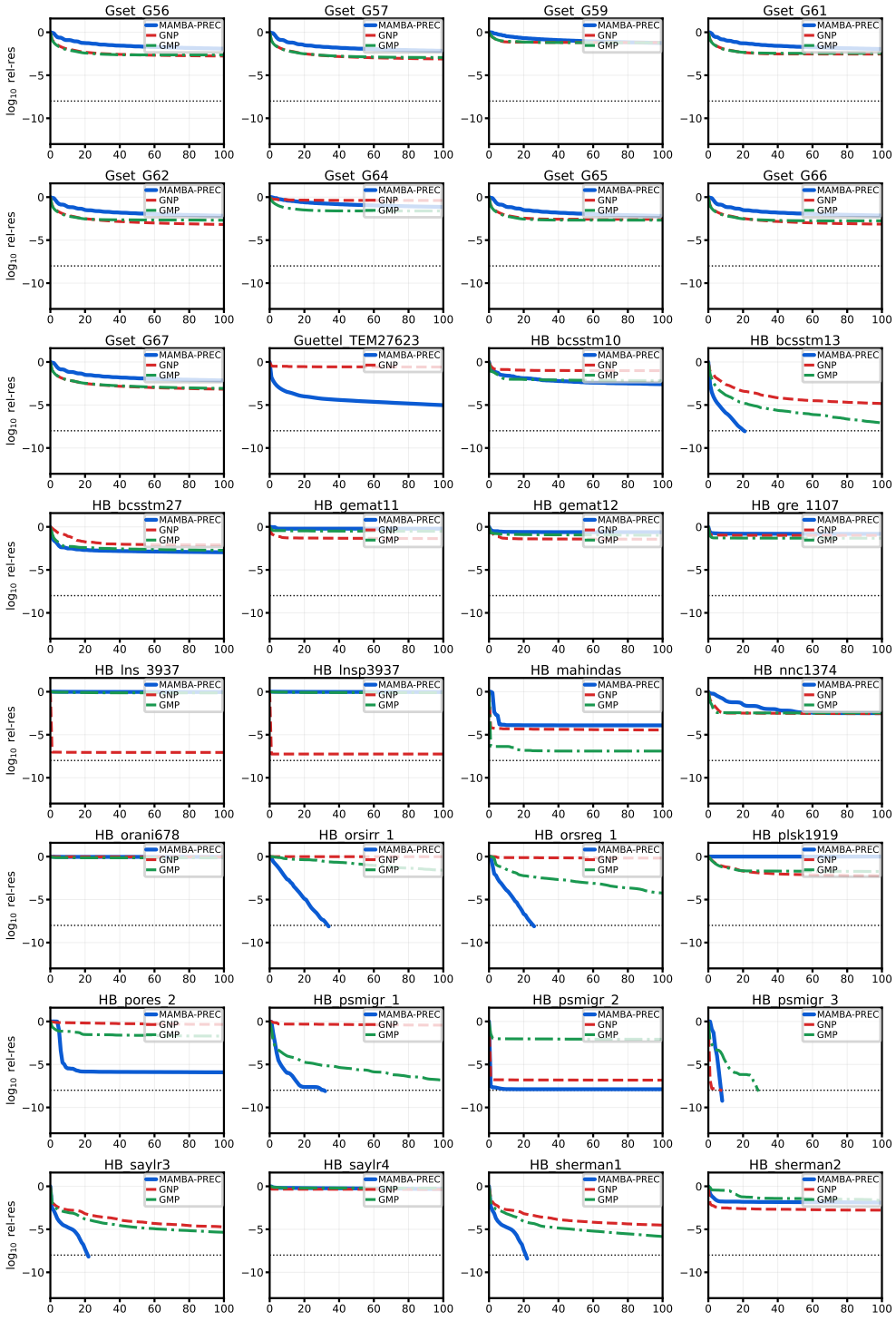

Cunningham_ga8fk
DIMACS10_ak2010
DIMACS10_ct2010
DIMACS10_de2010
DIMACS10_hi2010
DIMACS10_me2010
DIMACS10_nh2010
DIMACS10_nv2010
DIMACS10_ri2010
DIMACS10_sd2010
DIMACS10_vt2010
DIMACS10_wy2010
DRIVCAV_cavity05
DRIVCAV_cavity06
DRIVCAV_cavity07
DRIVCAV_cavity08
DRIVCAV_cavity09
DRIVCAV_cavity10
DRIVCAV_cavity11
DRIVCAV_cavity12
DRIVCAV_cavity13
DRIVCAV_cavity14
DRIVCAV_cavity15
DRIVCAV_cavity16
DRIVCAV_cavity17
DRIVCAV_cavity18
DRIVCAV_cavity19
DRIVCAV_cavity20
DRIVCAV_cavity21
DRIVCAV_cavity22
DRIVCAV_cavity23
DRIVCAV_cavity24
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

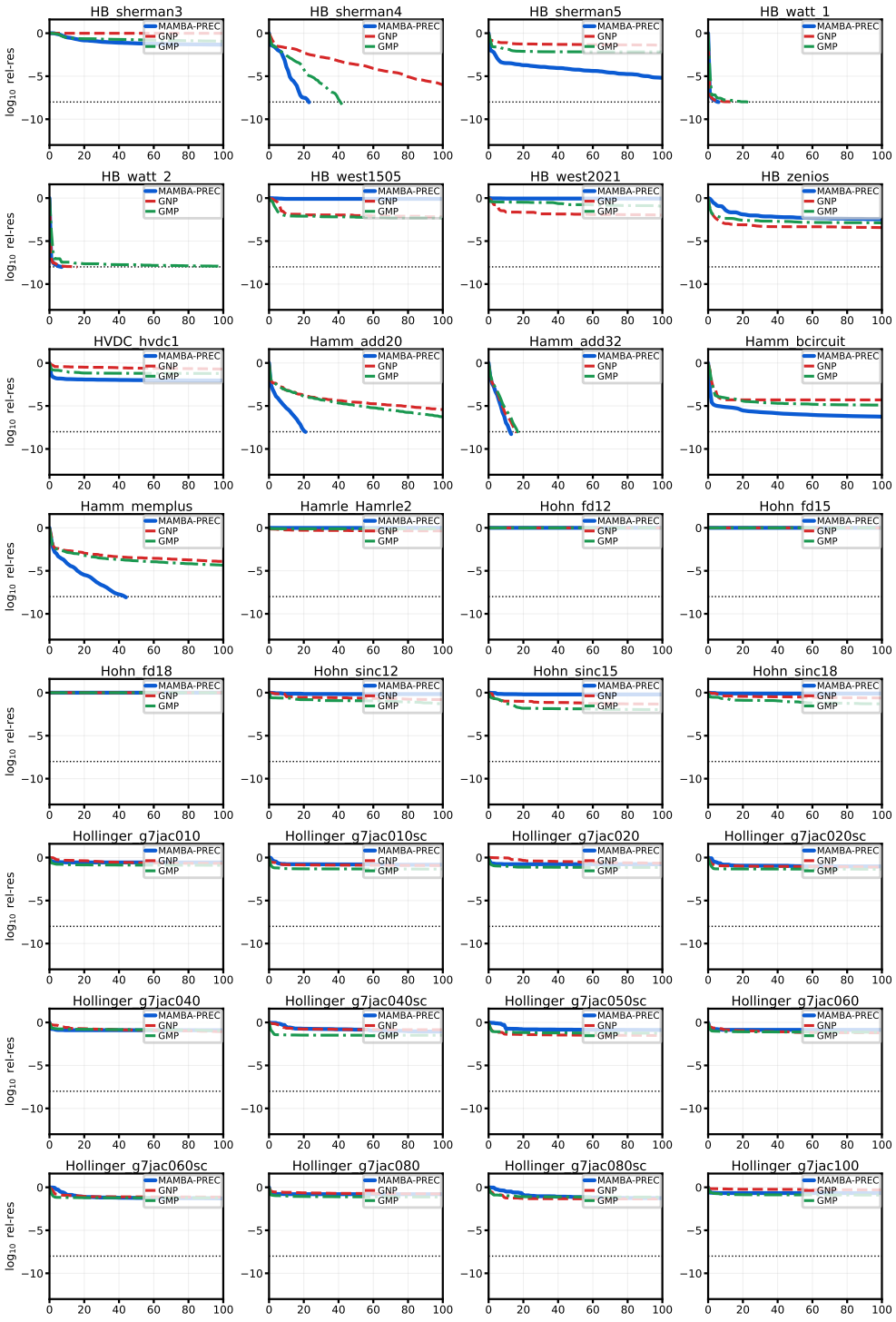

DRIVCAV_cavity25
DRIVCAV_cavity26
Engwirda_airfoil_2d
FEMLAB_ns3Da
FEMLAB_poisson3Da
FEMLAB_sme3Da
FIDAP_ex11
FIDAP_ex12
FIDAP_ex14
FIDAP_ex18
FIDAP_ex19
FIDAP_ex20
FIDAP_ex23
FIDAP_ex24
FIDAP_ex26
FIDAP_ex28
FIDAP_ex29
FIDAP_ex31
FIDAP_ex32
FIDAP_ex35
FIDAP_ex36
FIDAP_ex37
FIDAP_ex4
FIDAP_ex40
FIDAP_ex6
FIDAP_ex7
FIDAP_ex8
FlowIPM22_uni_chimera_i3
FlowIPM22_uni_chimera_i4
Fluorem_GT01R
GHS_indef_a0nsdsil
GHS_indef_a2nnsnsl
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

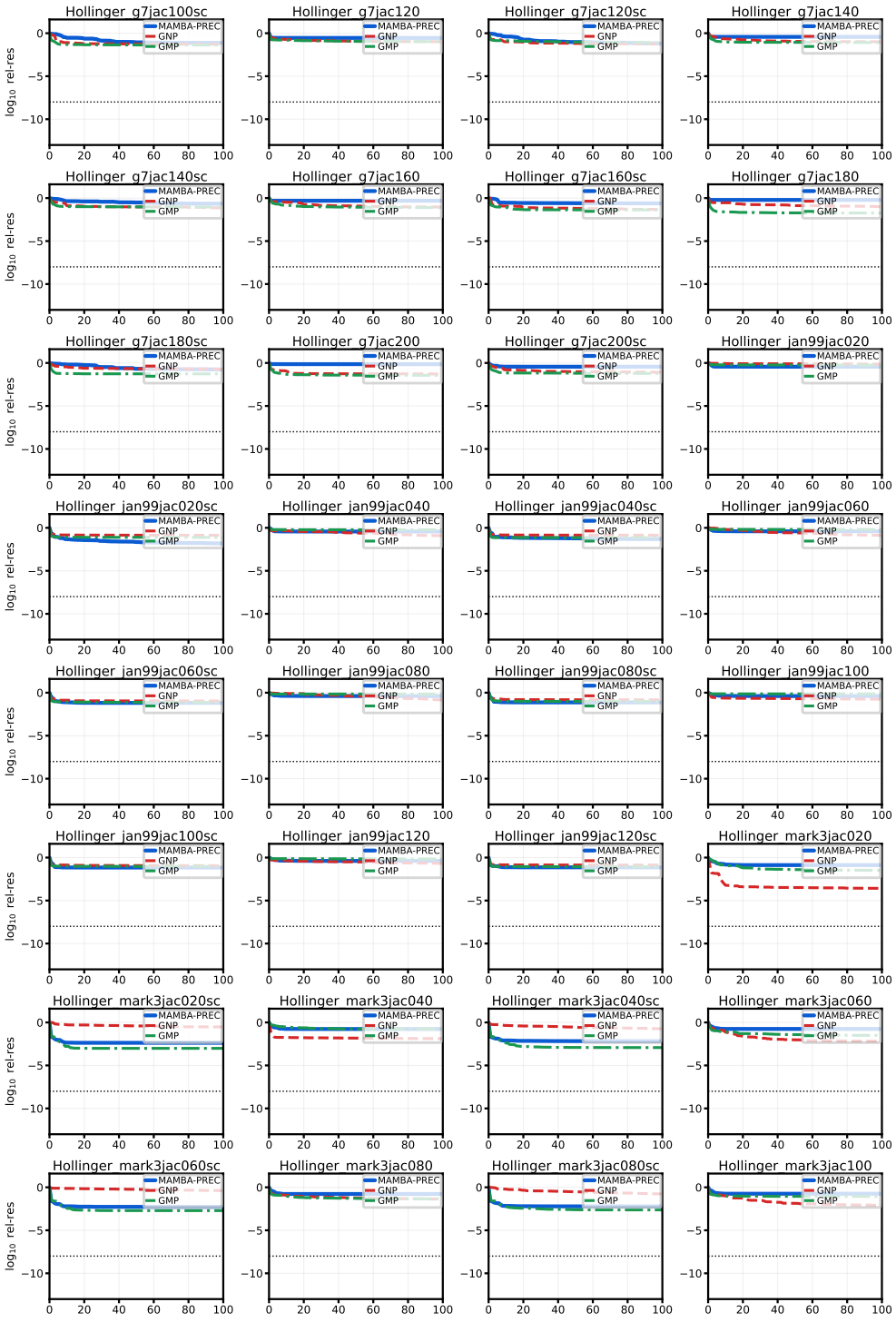

GHS_indef_a5esindl
GHS_indef_aug2d
GHS_indef_aug2dc
GHS_indef_aug3d
GHS_indef_aug3dcqp
GHS_indef_blockqp1
GHS_indef_bloweya
GHS_indef_bloweybl
GHS_indef_boyd1
GHS_indef_brainpc2
GHS_indef_bratu3d
GHS_indef_c-55
GHS_indef_c-58
GHS_indef_c-59
GHS_indef_c-62ghs
GHS_indef_c-63
GHS_indef_c-68
GHS_indef_c-69
GHS_indef_c-70
GHS_indef_c-71
GHS_indef_c-72
GHS_indef_cont-201
GHS_indef_copter2
GHS_indef_cvxqp3
GHS_indef_dawson5
GHS_indef_dixmaanl
GHS_indef_dtoc
GHS_indef_helm3d01
GHS_indef_k1_san
GHS_indef_laser
GHS_indef_linverse
GHS_indef_mario001
MAMBA-PREC
GNP
GMP
log10 rel-res

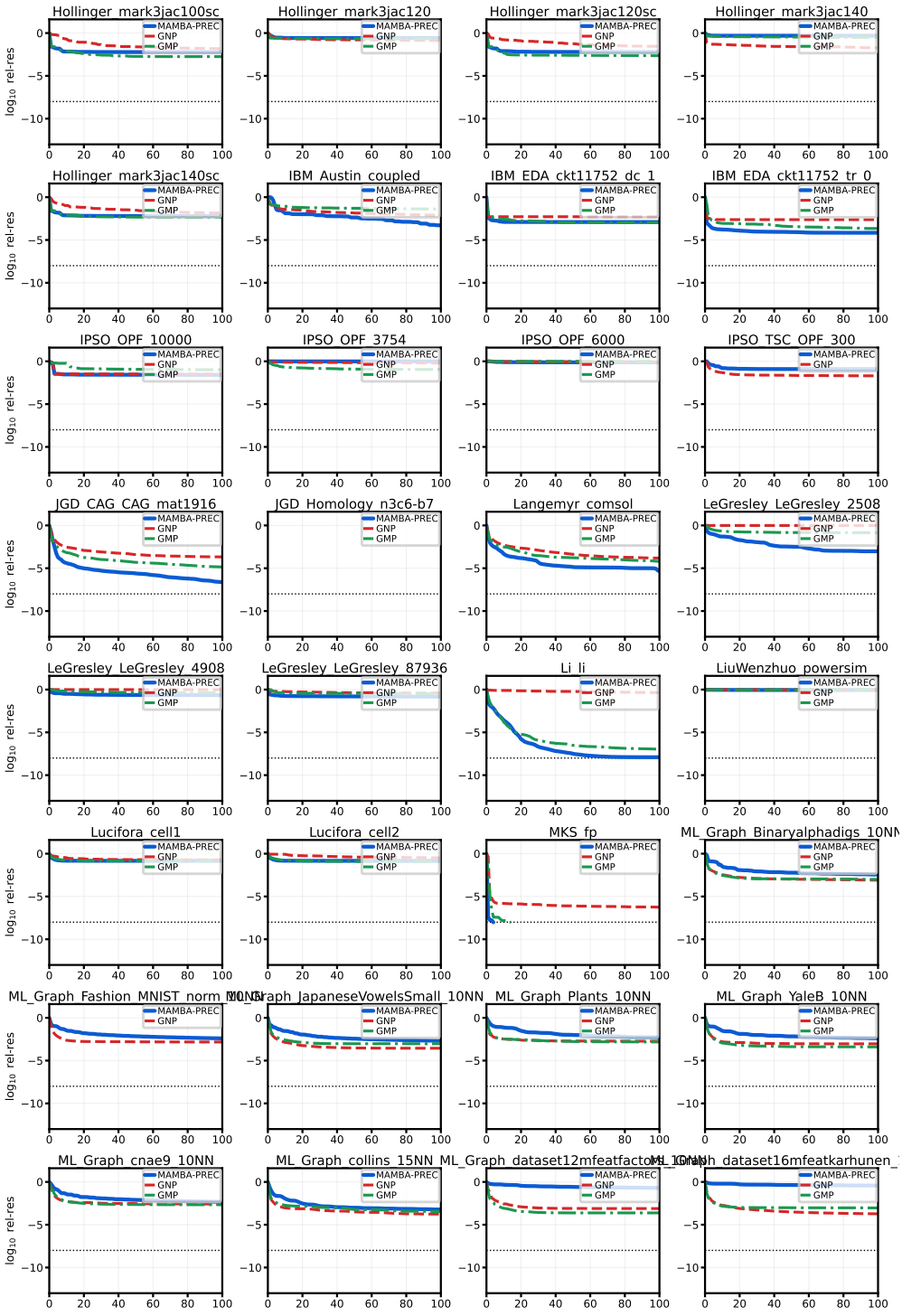

GHS_indef_ncvxbqp1
GHS_indef_ncvxqp1
GHS_indef_ncvxqp3
GHS_indef_ncvxqp5
GHS_indef_ncvxqp7
GHS_indef_ncvxqp9
GHS_indef_olesnik0
GHS_indef_qpband
GHS_indef_sit100
GHS_indef_sparsine
GHS_indef_spmsrtls
GHS_indef_stokes128
GHS_indef_stokes64
GHS_indef_stokes64s
GHS_indef_tuma1
GHS_indef_tuma2
Gaertner_big
Gaertner_nopoly
Gaertner_pesa
Garon_garon1
Garon_garon2
Goodwin_Goodwin_010
Goodwin_Goodwin_013
Goodwin_Goodwin_017
Goodwin_Goodwin_023
Goodwin_Goodwin_030
Goodwin_Goodwin_040
Goodwin_Goodwin_054
Goodwin_Goodwin_071
Goodwin_goodwin
Goodwin_rim
Graham_graham1
MAMBA-PREC
GNP
GMP
log10 rel-res

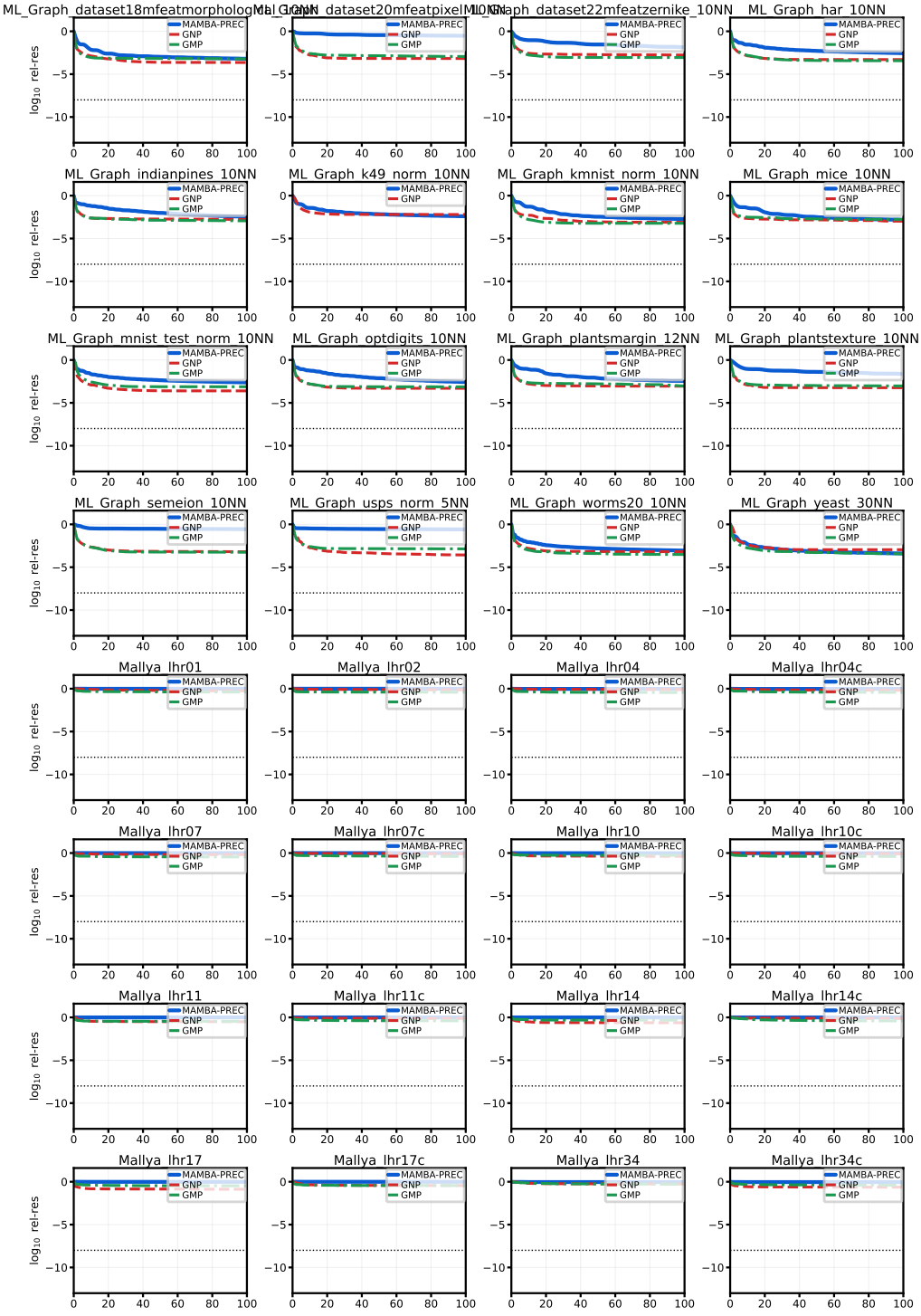

Grueninger_windtunnel_evap2d
Grueninger_windtunnel_evap3d
Grund_b2_ss
Grund_b_dyn
Grund_bayer01
Grund_bayer02
Grund_bayer03
Grund_bayer04
Grund_bayer05
Grund_bayer06
Grund_bayer07
Grund_bayer08
Grund_bayer09
Grund_bayer10
Grund_meg1
Grund_meg4
Grund_poli
Grund_poli3
Grund_poli4
Grund_poli_large
Gset_G27
Gset_G28
Gset_G29
Gset_G30
Gset_G31
Gset_G32
Gset_G33
Gset_G34
Gset_G39
Gset_G40
Gset_G41
Gset_G42
MAMBA-PREC
GNP
GMP
log10 rel-res

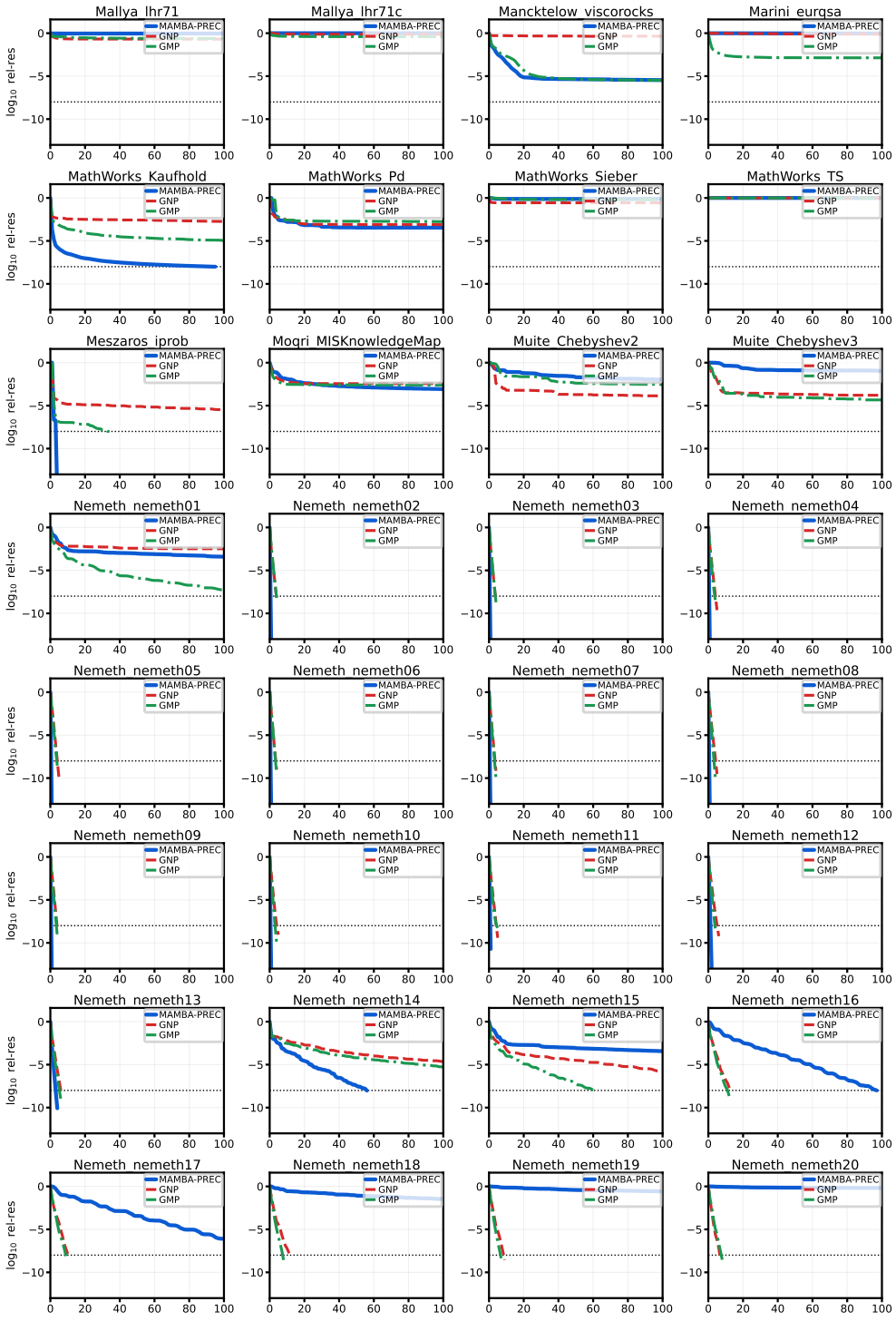

Gset_G56
Gset_G57
Gset_G59
Gset_G61
Gset_G62
Gset_G64
Gset_G65
Gset_G66
Gset_G67
Guettel_TEM27623
HB_bcsstm10
HB_bcsstm13
HB_bcsstm27
HB_gemat11
HB_gemat12
HB_gre_1107
HB_lns_3937
HB_lnsp3937
HB_mahindas
HB_nnc1374
HB_orani678
HB_orsirr_1
HB_orsreg_1
HB_plsk1919
HB_pores_2
HB_psmigr_1
HB_psmigr_2
HB_psmigr_3
HB_saylr3
HB_saylr4
HB_sherman1
HB_sherman2
MAMBA-PREC
GNP
GMP
log10 rel-res

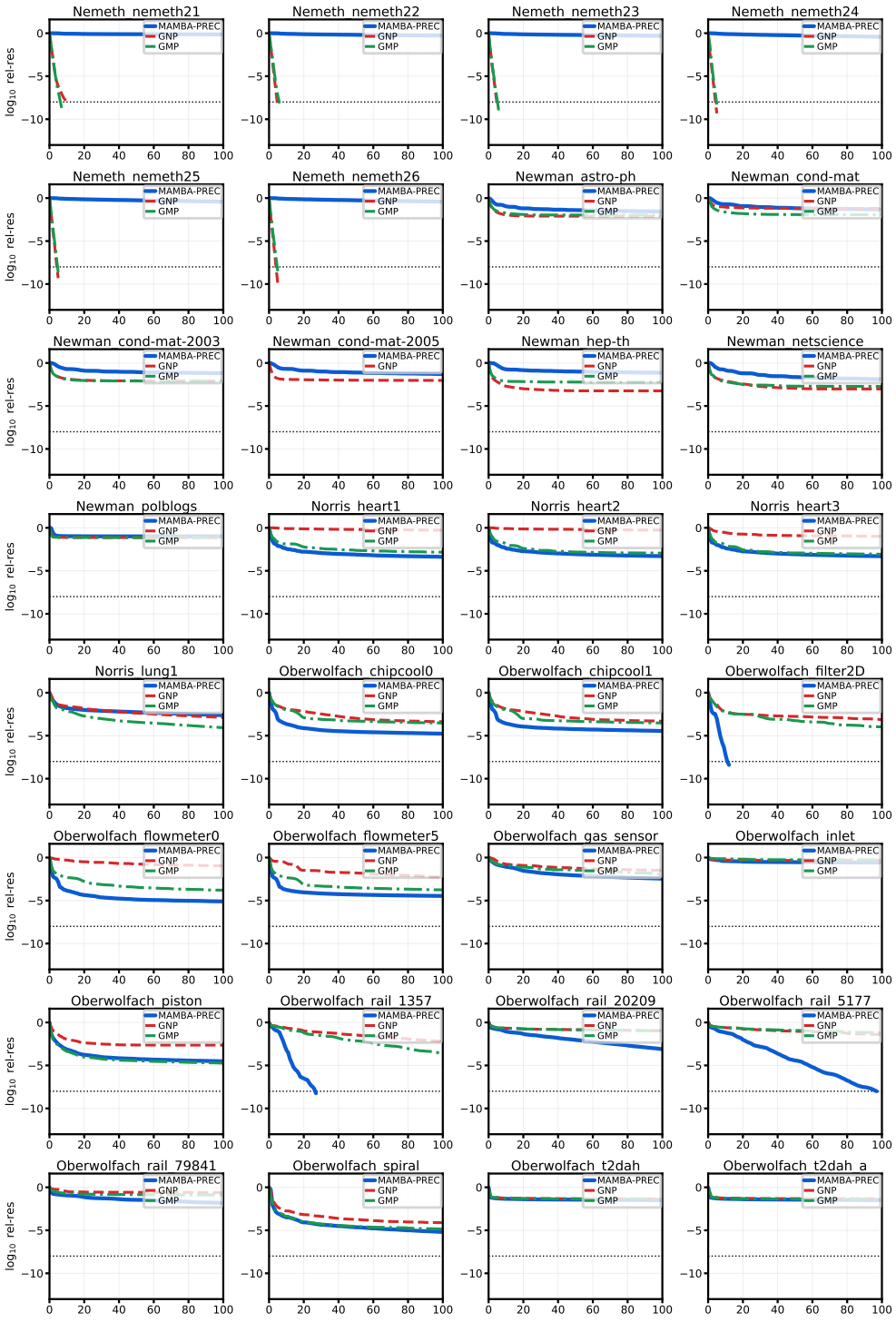

HB_sherman3
HB_sherman4
HB_sherman5
HB_watt_1
HB_watt_2
HB_west1505
HB_west2021
HB_zenios
HVDC_hvdc1
Hamm_add20
Hamm_add32
Hamm_bcircuit
Hamm_memplus
Hamrle_Hamrle2
Hohn_fd12
Hohn_fd15
Hohn_fd18
Hohn_sinc12
Hohn_sinc15
Hohn_sinc18
Hollinger_g7jac010
Hollinger_g7jac010sc
Hollinger_g7jac020
Hollinger_g7jac020sc
Hollinger_g7jac040
Hollinger_g7jac040sc
Hollinger_g7jac050sc
Hollinger_g7jac060
Hollinger_g7jac060sc
Hollinger_g7jac080
Hollinger_g7jac080sc
Hollinger_g7jac100
MAMBA-PREC
GNP
GMP
log10 rel-res

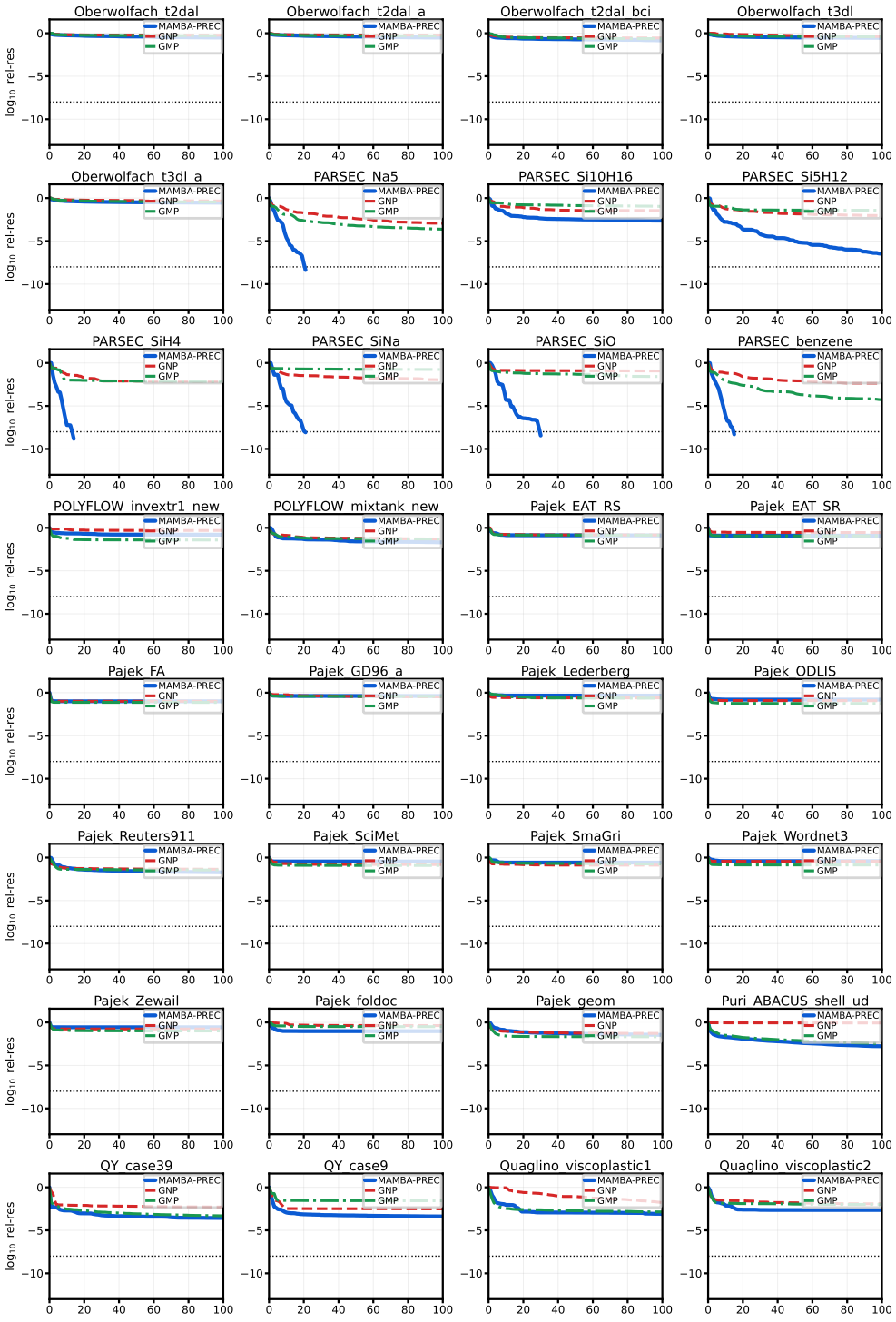

Hollinger_g7jac100sc
Hollinger_g7jac120
Hollinger_g7jac120sc
Hollinger_g7jac140
Hollinger_g7jac140sc
Hollinger_g7jac160
Hollinger_g7jac160sc
Hollinger_g7jac180
Hollinger_g7jac180sc
Hollinger_g7jac200
Hollinger_g7jac200sc
Hollinger_jan99jac020
Hollinger_jan99jac020sc
Hollinger_jan99jac040
Hollinger_jan99jac040sc
Hollinger_jan99jac060
Hollinger_jan99jac060sc
Hollinger_jan99jac080
Hollinger_jan99jac080sc
Hollinger_jan99jac100
Hollinger_jan99jac100sc
Hollinger_jan99jac120
Hollinger_jan99jac120sc
Hollinger_mark3jac020
Hollinger_mark3jac020sc
Hollinger_mark3jac040
Hollinger_mark3jac040sc
Hollinger_mark3jac060
Hollinger_mark3jac060sc
Hollinger_mark3jac080
Hollinger_mark3jac080sc
Hollinger_mark3jac100
MAMBA-PREC
GNP
GMP
log10 rel-res

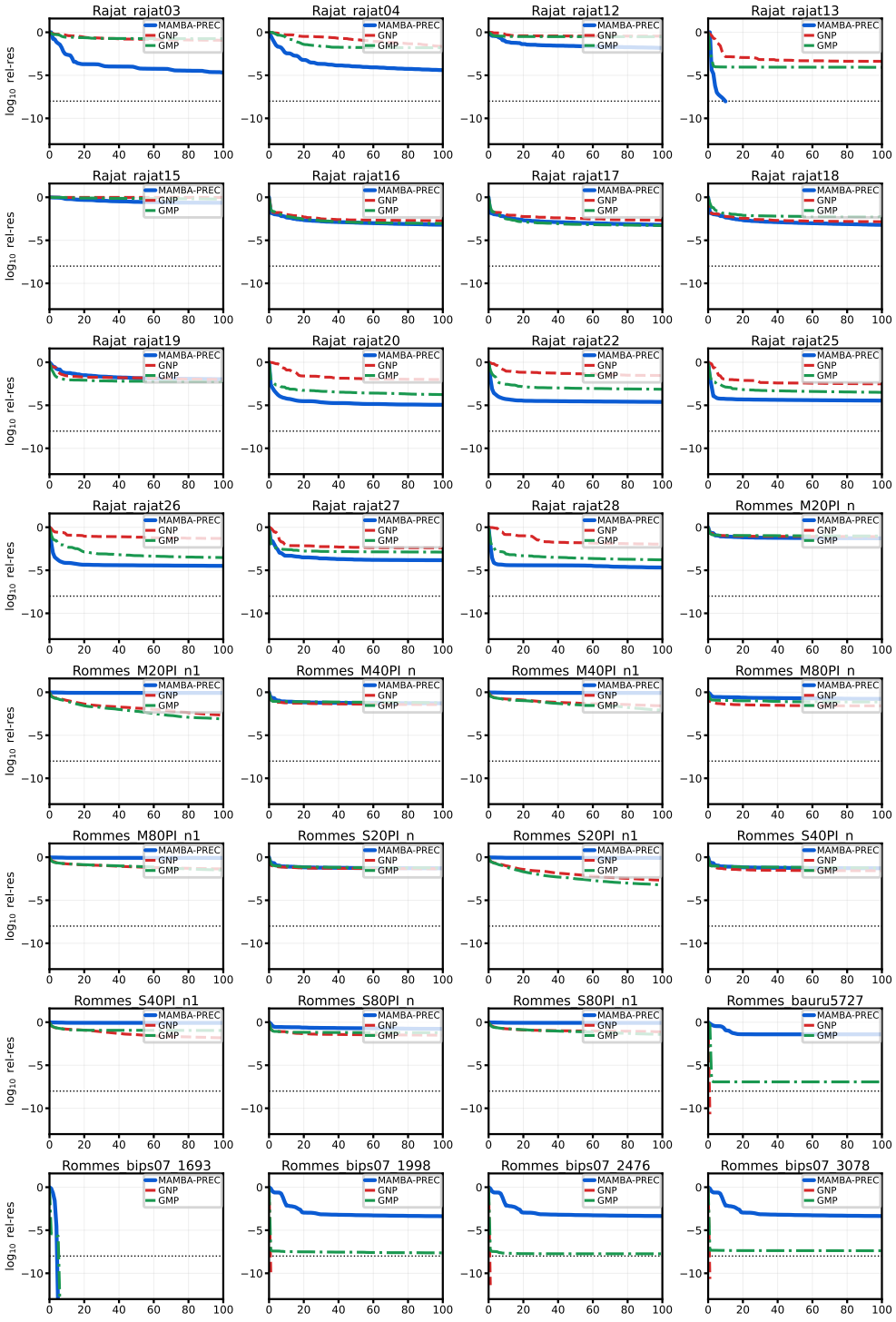

Hollinger_mark3jac100sc
Hollinger_mark3jac120
Hollinger_mark3jac120sc
Hollinger_mark3jac140
Hollinger_mark3jac140sc
IBM_Austin_coupled
IBM_EDA_ckt11752_dc_1
IBM_EDA_ckt11752_tr_0
IPSO_OPF_10000
IPSO_OPF_3754
IPSO_OPF_6000
IPSO_TSC_OPF_300
JGD_CAG_CAG_mat1916
JGD_Homology_n3c6-b7
Langemyr_comsol
LeGresley_LeGresley_2508
LeGresley_LeGresley_4908
LeGresley_LeGresley_87936
Li_li
LiuWenzhuo_powersim
Lucifora_cell1
Lucifora_cell2
MKS_fp
ML_Graph_Binaryalphadigs_10NN
ML_Graph_Fashion_MNIST_norm_10NN
ML_Graph_JapaneseVowelsSmall_10NN
ML_Graph_Plants_10NN
ML_Graph_YaleB_10NN
ML_Graph_cnae9_10NN
ML_Graph_collins_15NN
ML_Graph_dataset12mfeatfactors_10NN
ML_Graph_dataset16mfeatkarhunen_10NN
MAMBA-PREC
GNP
GMP
log10 rel-res

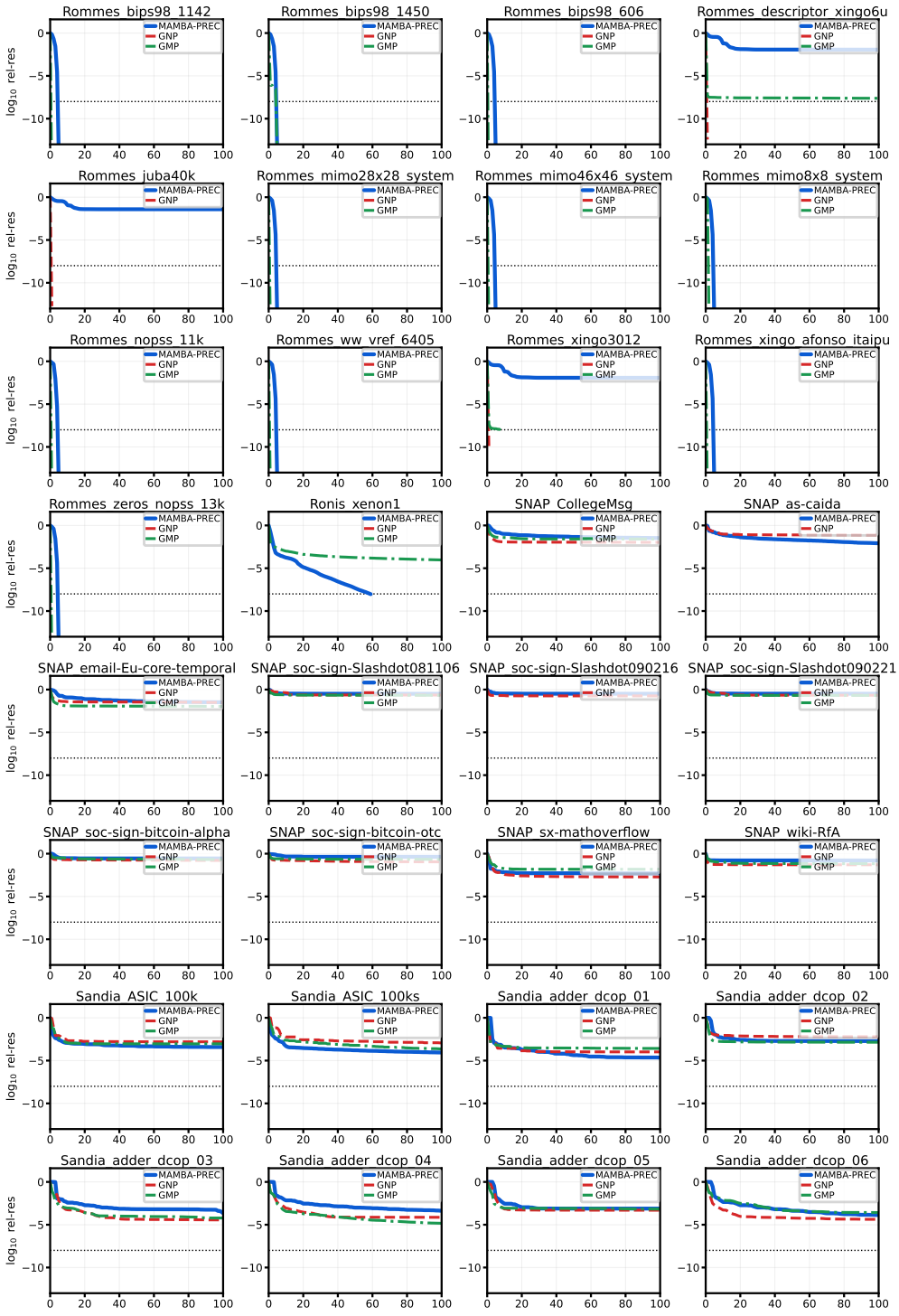

ML_Graph_dataset18mfeatmorphologic_10NN
ML_Graph_dataset20mfeatpixel_10NN
ML_Graph_dataset22mfeatzernike_10NN
ML_Graph_har_10NN
ML_Graph_indianpines_10NN
ML_Graph_k49_norm_10NN
ML_Graph_kmnist_norm_10NN
ML_Graph_mice_10NN
ML_Graph_mnist_test_norm_10NN
ML_Graph_optdigits_10NN
ML_Graph_plantsmargin_12NN
ML_Graph_plantstexture_10NN
ML_Graph_semeion_10NN
ML_Graph_usps_norm_5NN
ML_Graph_worms20_10NN
ML_Graph_yeast_30NN
Mallya_lhr01
Mallya_lhr02
Mallya_lhr04
Mallya_lhr04c
Mallya_lhr07
Mallya_lhr07c
Mallya_lhr10
Mallya_lhr10c
Mallya_lhr11
Mallya_lhr11c
Mallya_lhr14
Mallya_lhr14c
Mallya_lhr17
Mallya_lhr17c
Mallya_lhr34
Mallya_lhr34c
MAMBA-PREC
GNP
GMP
log10 rel-res

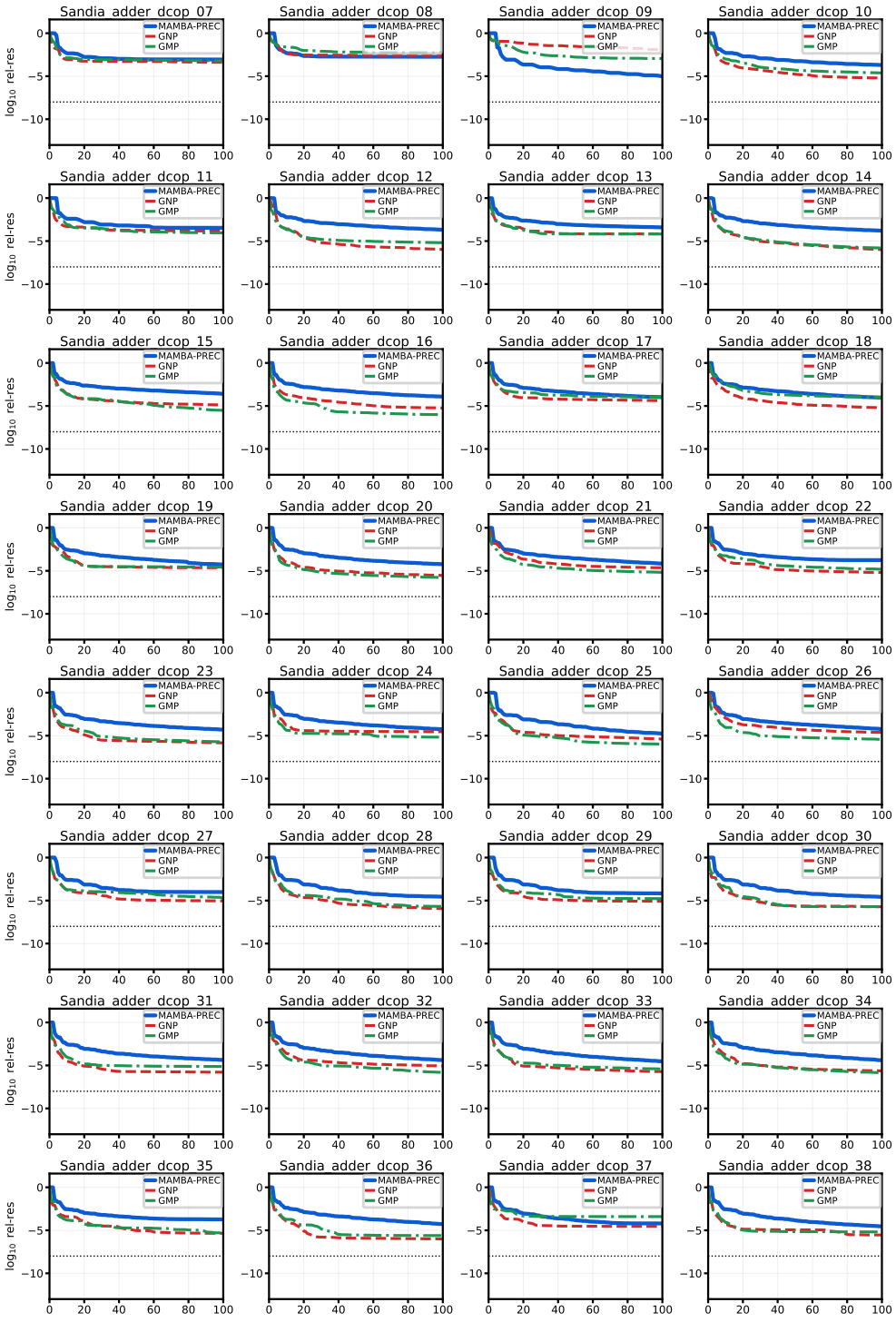

Mallya_lhr71
Mallya_lhr71c
Mancktelow_viscorocks
Marini_eurgsa
MathWorks_Kaufhold
MathWorks_Pd
MathWorks_Sieber
MathWorks_TS
Meszaros_iprob
Mogri_MISKnowledgeMap
Muite_Chebyshev2
Muite_Chebyshev3
Nemeth_nemeth01
Nemeth_nemeth02
Nemeth_nemeth03
Nemeth_nemeth04
Nemeth_nemeth05
Nemeth_nemeth06
Nemeth_nemeth07
Nemeth_nemeth08
Nemeth_nemeth09
Nemeth_nemeth10
Nemeth_nemeth11
Nemeth_nemeth12
Nemeth_nemeth13
Nemeth_nemeth14
Nemeth_nemeth15
Nemeth_nemeth16
Nemeth_nemeth17
Nemeth_nemeth18
Nemeth_nemeth19
Nemeth_nemeth20
MAMBA-PREC
GNP
GMP
log10 rel-res

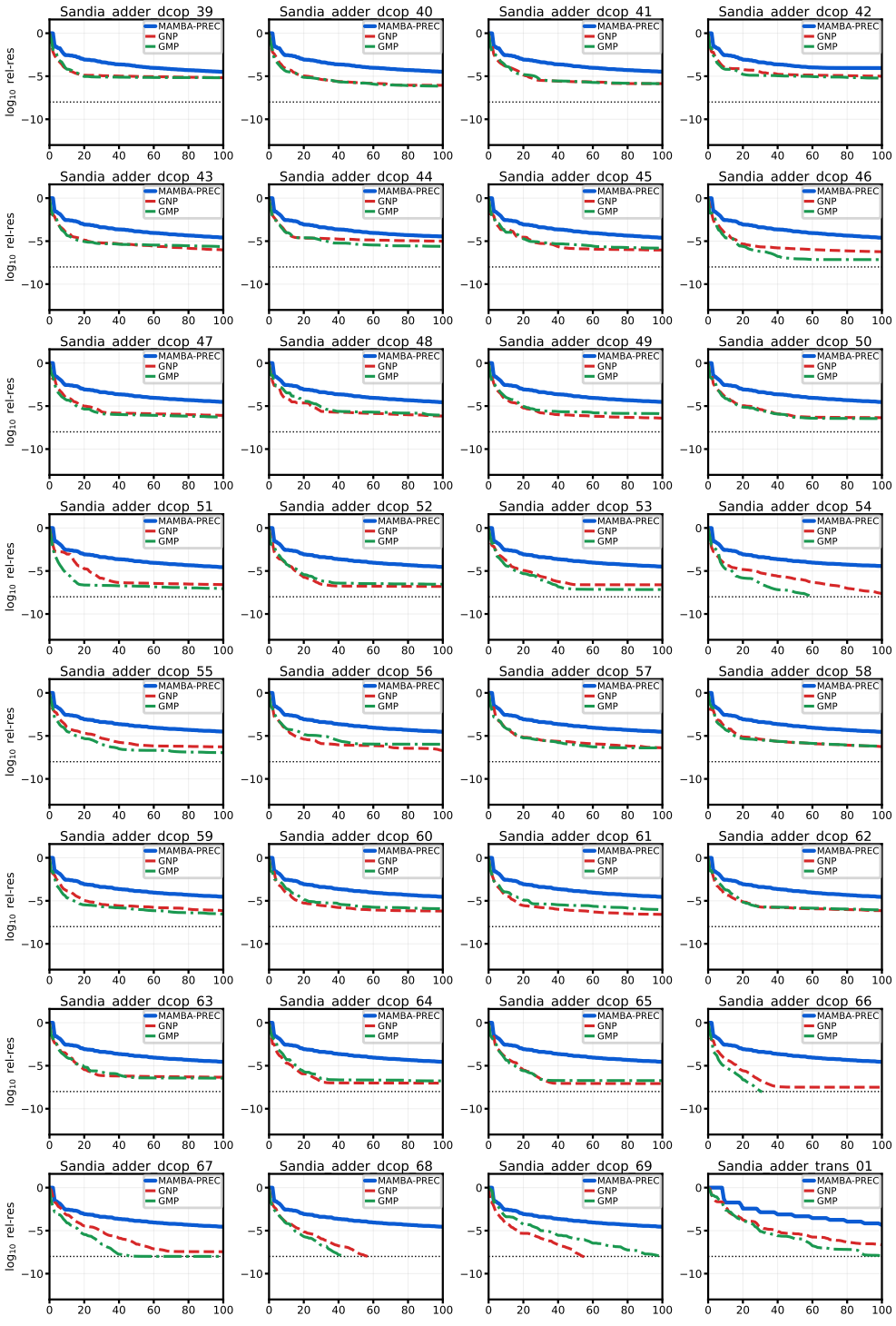

Nemeth_nemeth21
Nemeth_nemeth22
Nemeth_nemeth23
Nemeth_nemeth24
Nemeth_nemeth25
Nemeth_nemeth26
Newman_astro-ph
Newman_cond-mat
Newman_cond-mat-2003
Newman_cond-mat-2005
Newman_hep-th
Newman_netscience
Newman_polblogs
Norris_heart1
Norris_heart2
Norris_heart3
Norris_lung1
Oberwolfach_chipcool0
Oberwolfach_chipcool1
Oberwolfach_filter2D
Oberwolfach_flowmeter0
Oberwolfach_flowmeter5
Oberwolfach_gas_sensor
Oberwolfach_inlet
Oberwolfach_piston
Oberwolfach_rail_1357
Oberwolfach_rail_20209
Oberwolfach_rail_5177
Oberwolfach_rail_79841
Oberwolfach_spiral
Oberwolfach_t2dah
Oberwolfach_t2dah_a
MAMBA-PREC
GNP
GMP
log10 rel-res

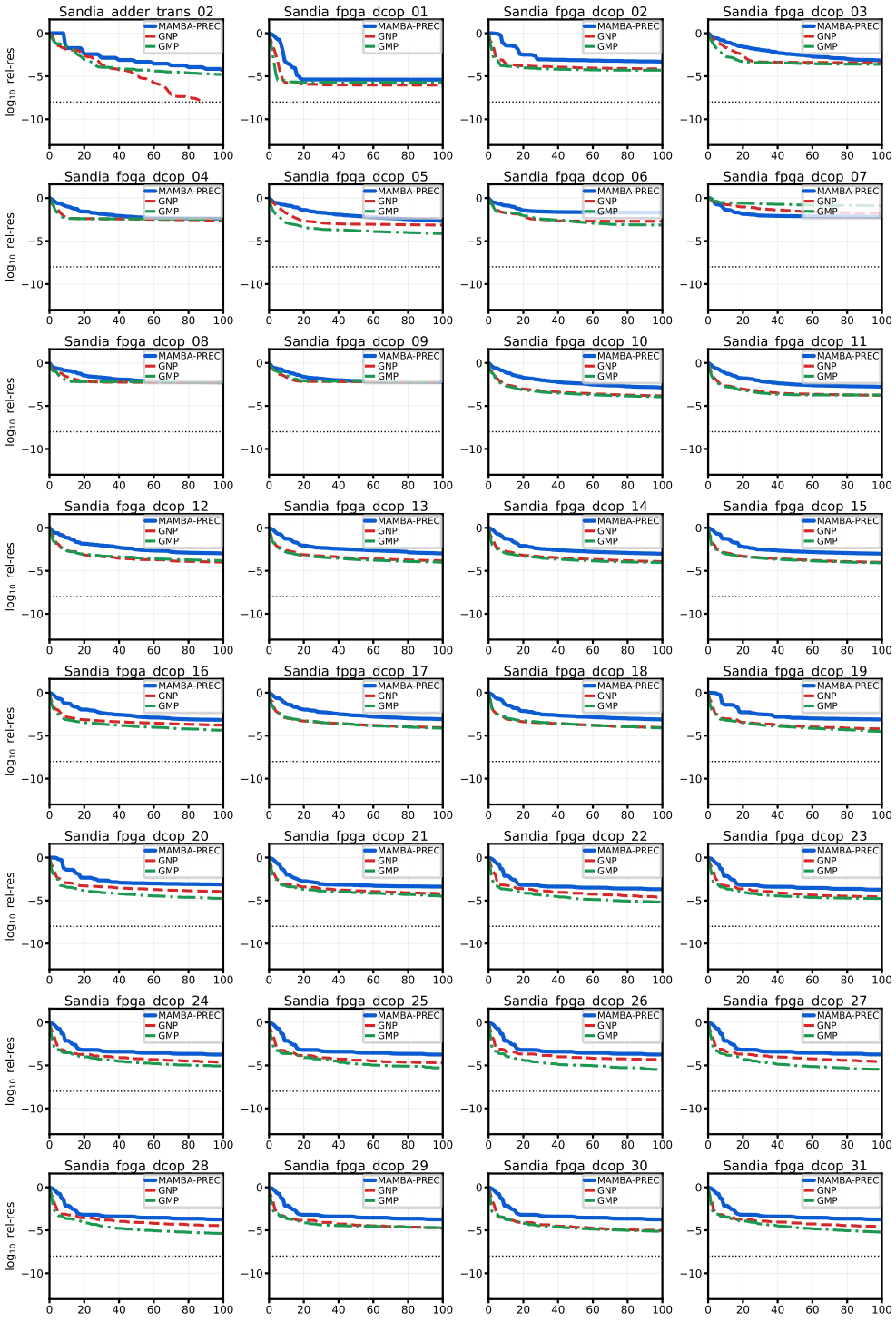

Oberwolfach_t2dal
Oberwolfach_t2dal_a
Oberwolfach_t2dal_bci
Oberwolfach_t3dl
Oberwolfach_t3dl_a
PARSEC_Na5
PARSEC_Si10H16
PARSEC_Si5H12
PARSEC_SiH4
PARSEC_SiNa
PARSEC_SiO
PARSEC_benzene
POLYFLOW_invextr1_new
POLYFLOW_mixtank_new
Pajek_EAT_RS
Pajek_EAT_SR
Pajek_FA
Pajek_GD96_a
Pajek_Lederberg
Pajek_ODLIS
Pajek_Reuters911
Pajek_SciMet
Pajek_SmaGri
Pajek_Wordnet3
Pajek_Zewail
Pajek_foldoc
Pajek_geom
Puri_ABACUS_shell_ud
QY_case39
QY_case9
Quaglino_viscoplastic1
Quaglino_viscoplastic2
MAMBA-PREC
GNP
GMP
log10 rel-res

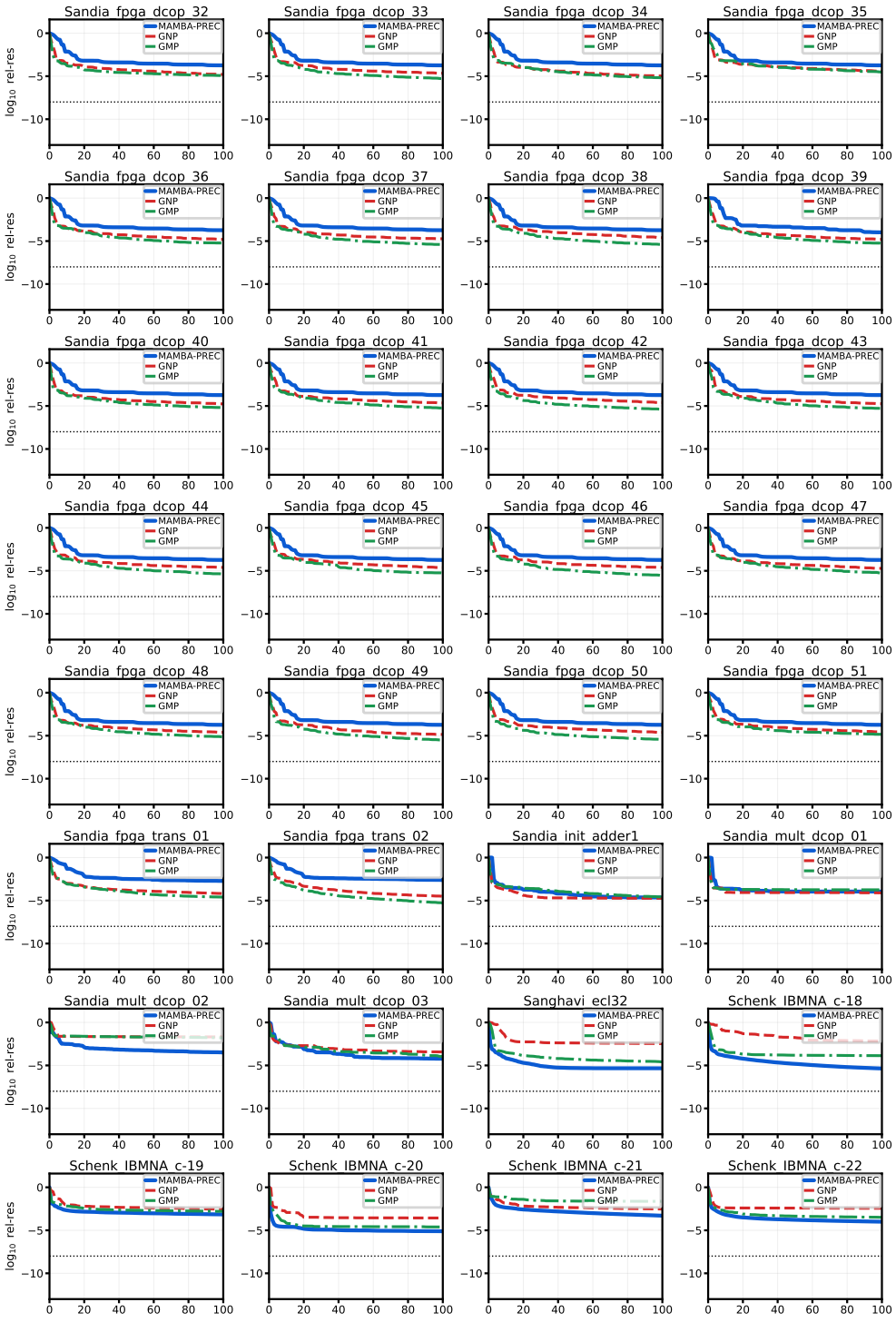

Rajat_rajat03
Rajat_rajat04
Rajat_rajat12
Rajat_rajat13
Rajat_rajat15
Rajat_rajat16
Rajat_rajat17
Rajat_rajat18
Rajat_rajat19
Rajat_rajat20
Rajat_rajat22
Rajat_rajat25
Rajat_rajat26
Rajat_rajat27
Rajat_rajat28
Rommes_M20PI_n
Rommes_M20PI_n1
Rommes_M40PI_n
Rommes_M40PI_n1
Rommes_M80PI_n
Rommes_M80PI_n1
Rommes_S20PI_n
Rommes_S20PI_n1
Rommes_S40PI_n
Rommes_S40PI_n1
Rommes_S80PI_n
Rommes_S80PI_n1
Rommes_bauru5727
Rommes_bips07_1693
Rommes_bips07_1998
Rommes_bips07_2476
Rommes_bips07_3078
MAMBA-PREC
GNP
GMP
log10 rel-res

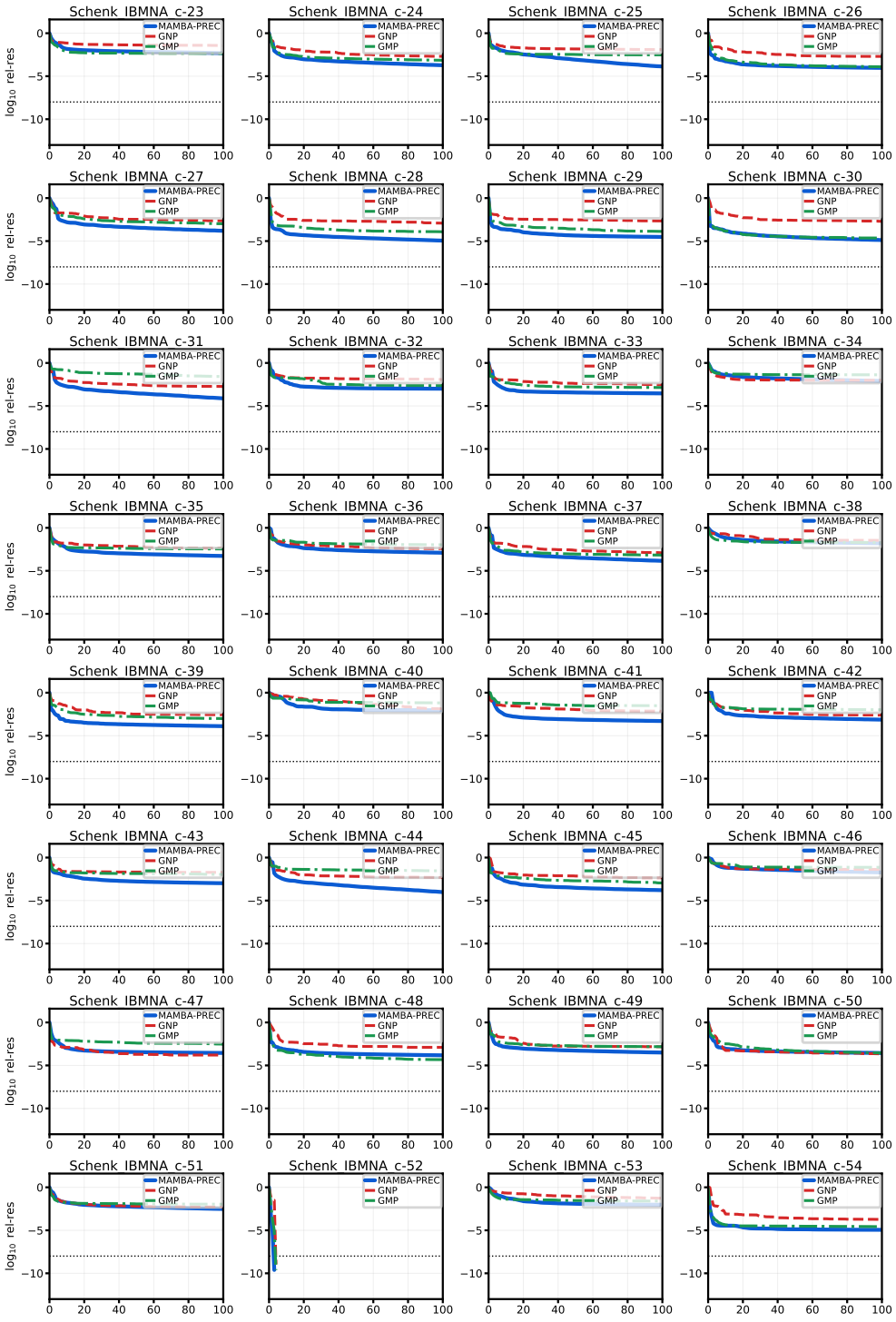

Rommes_bips98_1142
Rommes_bips98_1450
Rommes_bips98_606
Rommes_descriptor_xingo6u
Rommes_juba40k
Rommes_mimo28x28_system
Rommes_mimo46x46_system
Rommes_mimo8x8_system
Rommes_nopss_11k
Rommes_ww_vref_6405
Rommes_xingo3012
Rommes_xingo_afonso_itaipu
Rommes_zeros_nopss_13k
Ronis_xenon1
SNAP_CollegeMsg
SNAP_as-caida
SNAP_email-Eu-core-temporal
SNAP_soc-sign-Slashdot081106
SNAP_soc-sign-Slashdot090216
SNAP_soc-sign-Slashdot090221
SNAP_soc-sign-bitcoin-alpha
SNAP_soc-sign-bitcoin-otc
SNAP_sx-mathoverflow
SNAP_wiki-RfA
Sandia_ASIC_100k
Sandia_ASIC_100ks
Sandia_adder_dcop_01
Sandia_adder_dcop_02
Sandia_adder_dcop_03
Sandia_adder_dcop_04
Sandia_adder_dcop_05
Sandia_adder_dcop_06
MAMBA-PREC
GNP
GMP
log10 rel-res

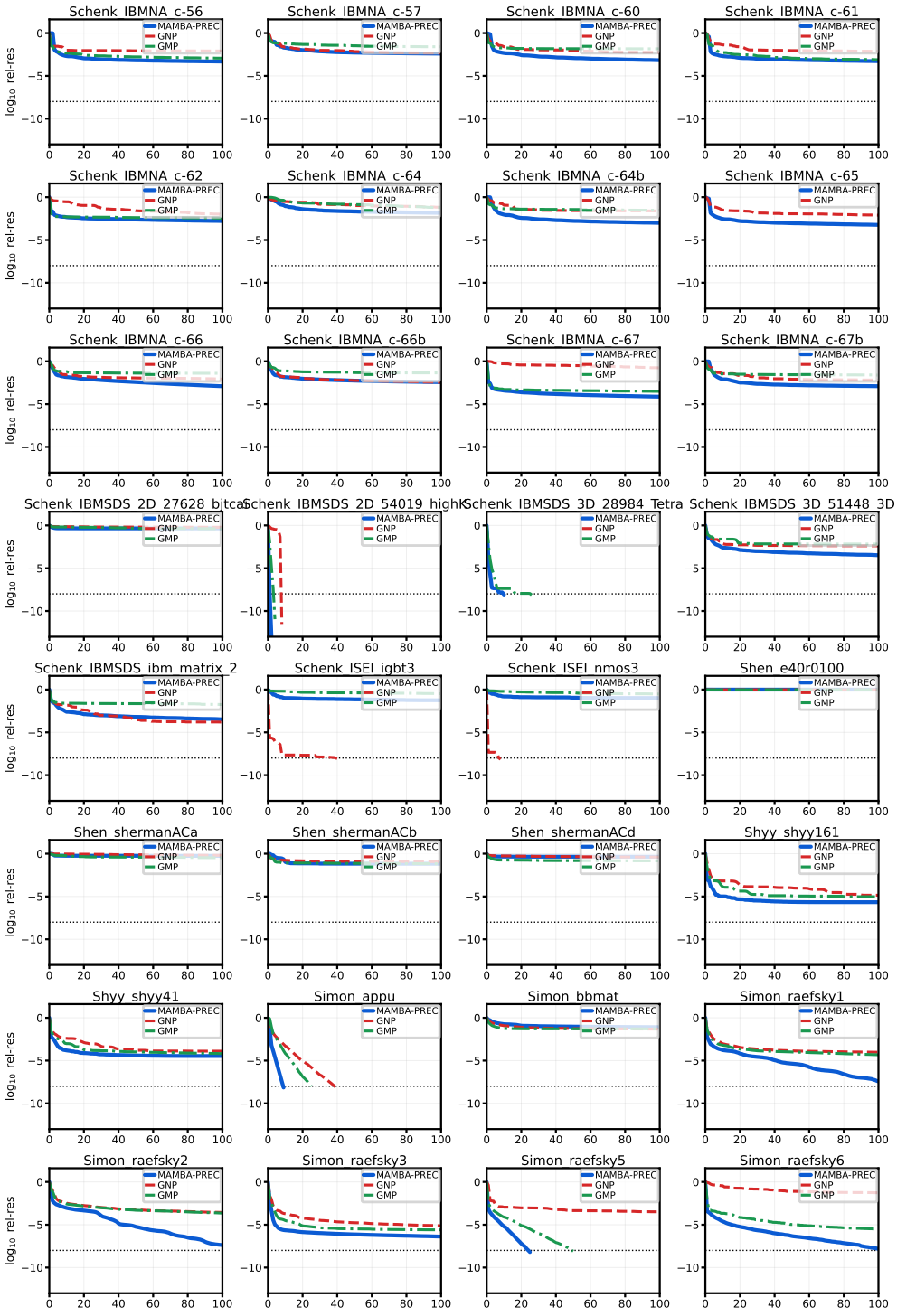

Sandia_adder_dcop_07
Sandia_adder_dcop_08
Sandia_adder_dcop_09
Sandia_adder_dcop_10
Sandia_adder_dcop_11
Sandia_adder_dcop_12
Sandia_adder_dcop_13
Sandia_adder_dcop_14
Sandia_adder_dcop_15
Sandia_adder_dcop_16
Sandia_adder_dcop_17
Sandia_adder_dcop_18
Sandia_adder_dcop_19
Sandia_adder_dcop_20
Sandia_adder_dcop_21
Sandia_adder_dcop_22
Sandia_adder_dcop_23
Sandia_adder_dcop_24
Sandia_adder_dcop_25
Sandia_adder_dcop_26
Sandia_adder_dcop_27
Sandia_adder_dcop_28
Sandia_adder_dcop_29
Sandia_adder_dcop_30
Sandia_adder_dcop_31
Sandia_adder_dcop_32
Sandia_adder_dcop_33
Sandia_adder_dcop_34
Sandia_adder_dcop_35
Sandia_adder_dcop_36
Sandia_adder_dcop_37
Sandia_adder_dcop_38
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

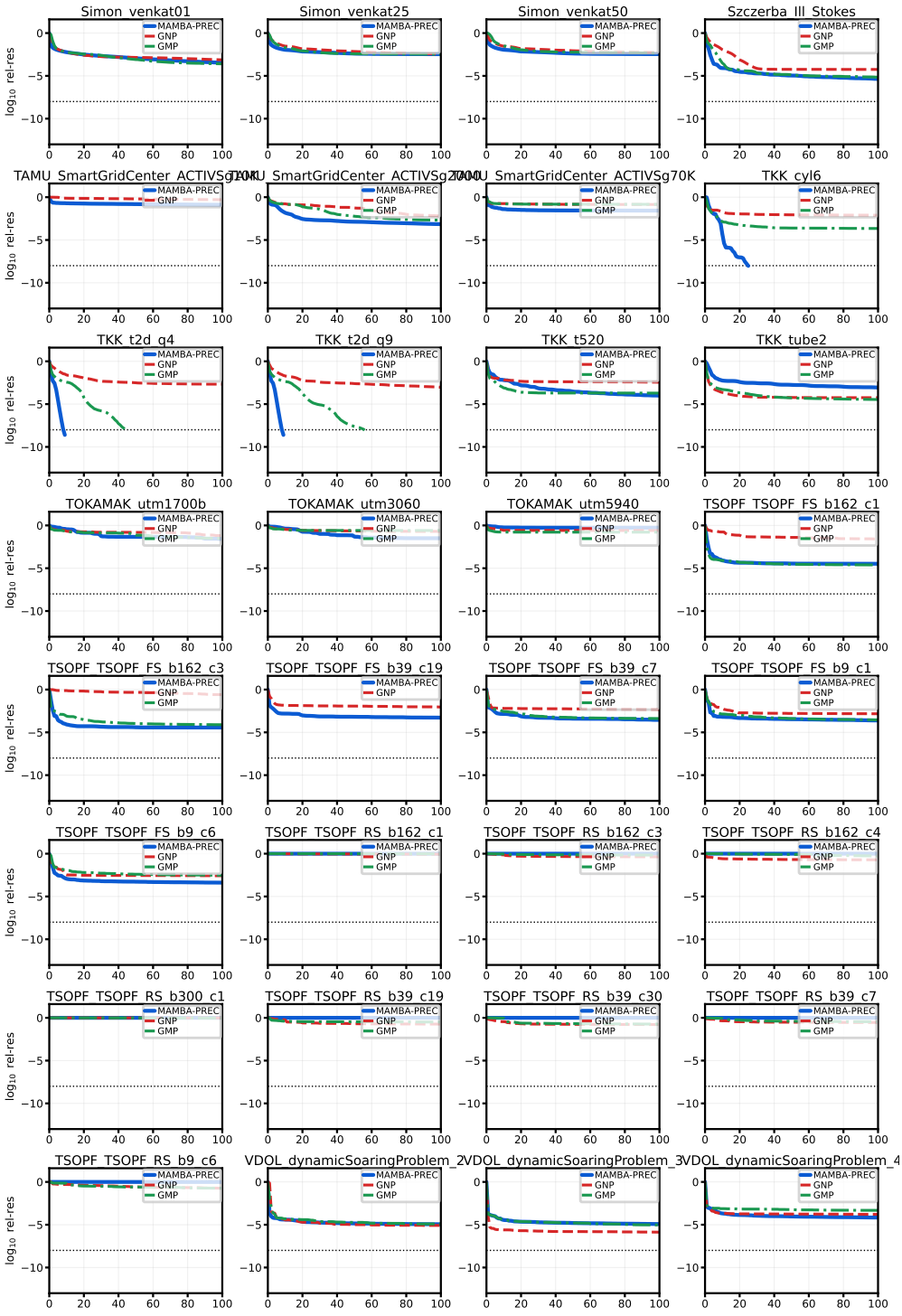

Sandia_adder_dcop_39
Sandia_adder_dcop_40
Sandia_adder_dcop_41
Sandia_adder_dcop_42
Sandia_adder_dcop_43
Sandia_adder_dcop_44
Sandia_adder_dcop_45
Sandia_adder_dcop_46
Sandia_adder_dcop_47
Sandia_adder_dcop_48
Sandia_adder_dcop_49
Sandia_adder_dcop_50
Sandia_adder_dcop_51
Sandia_adder_dcop_52
Sandia_adder_dcop_53
Sandia_adder_dcop_54
Sandia_adder_dcop_55
Sandia_adder_dcop_56
Sandia_adder_dcop_57
Sandia_adder_dcop_58
Sandia_adder_dcop_59
Sandia_adder_dcop_60
Sandia_adder_dcop_61
Sandia_adder_dcop_62
Sandia_adder_dcop_63
Sandia_adder_dcop_64
Sandia_adder_dcop_65
Sandia_adder_dcop_66
Sandia_adder_dcop_67
Sandia_adder_dcop_68
Sandia_adder_dcop_69
Sandia_adder_trans_01
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

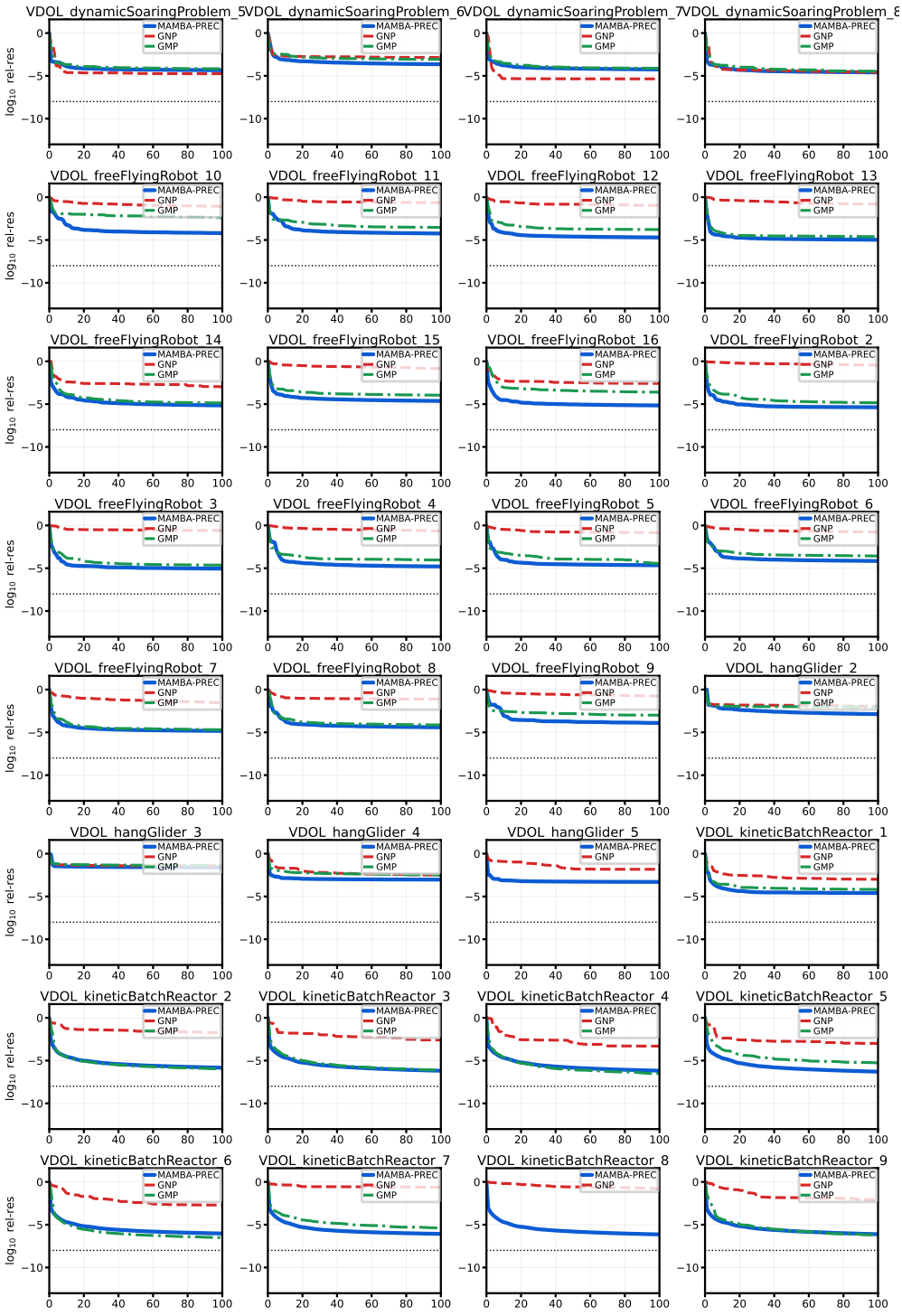

Sandia_adder_trans_02
Sandia_fpga_dcop_01
Sandia_fpga_dcop_02
Sandia_fpga_dcop_03
Sandia_fpga_dcop_04
Sandia_fpga_dcop_05
Sandia_fpga_dcop_06
Sandia_fpga_dcop_07
Sandia_fpga_dcop_08
Sandia_fpga_dcop_09
Sandia_fpga_dcop_10
Sandia_fpga_dcop_11
Sandia_fpga_dcop_12
Sandia_fpga_dcop_13
Sandia_fpga_dcop_14
Sandia_fpga_dcop_15
Sandia_fpga_dcop_16
Sandia_fpga_dcop_17
Sandia_fpga_dcop_18
Sandia_fpga_dcop_19
Sandia_fpga_dcop_20
Sandia_fpga_dcop_21
Sandia_fpga_dcop_22
Sandia_fpga_dcop_23
Sandia_fpga_dcop_24
Sandia_fpga_dcop_25
Sandia_fpga_dcop_26
Sandia_fpga_dcop_27
Sandia_fpga_dcop_28
Sandia_fpga_dcop_29
Sandia_fpga_dcop_30
Sandia_fpga_dcop_31
MAMBA-PREC
GNP
GMP
log10 rel-res

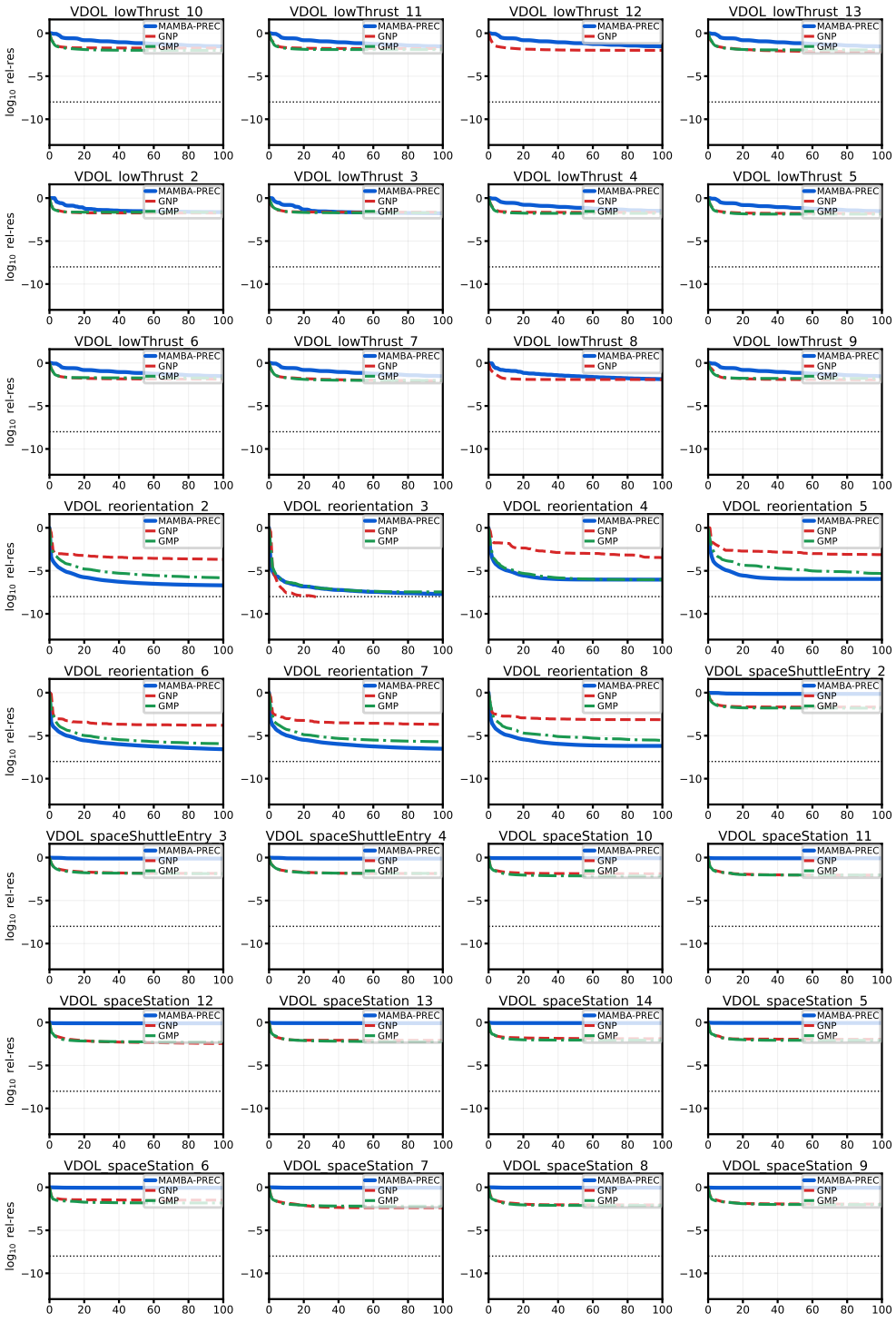

Sandia_fpga_dcop_32
Sandia_fpga_dcop_33
Sandia_fpga_dcop_34
Sandia_fpga_dcop_35
Sandia_fpga_dcop_36
Sandia_fpga_dcop_37
Sandia_fpga_dcop_38
Sandia_fpga_dcop_39
Sandia_fpga_dcop_40
Sandia_fpga_dcop_41
Sandia_fpga_dcop_42
Sandia_fpga_dcop_43
Sandia_fpga_dcop_44
Sandia_fpga_dcop_45
Sandia_fpga_dcop_46
Sandia_fpga_dcop_47
Sandia_fpga_dcop_48
Sandia_fpga_dcop_49
Sandia_fpga_dcop_50
Sandia_fpga_dcop_51
Sandia_fpga_trans_01
Sandia_fpga_trans_02
Sandia_init_adder1
Sandia_mult_dcop_01
Sandia_mult_dcop_02
Sandia_mult_dcop_03
Sanghavi_ecl32
Schenk_IBMNA_c-18
Schenk_IBMNA_c-19
Schenk_IBMNA_c-20
Schenk_IBMNA_c-21
Schenk_IBMNA_c-22
MAMBA-PREC
GNP
GMP
log10 rel-res

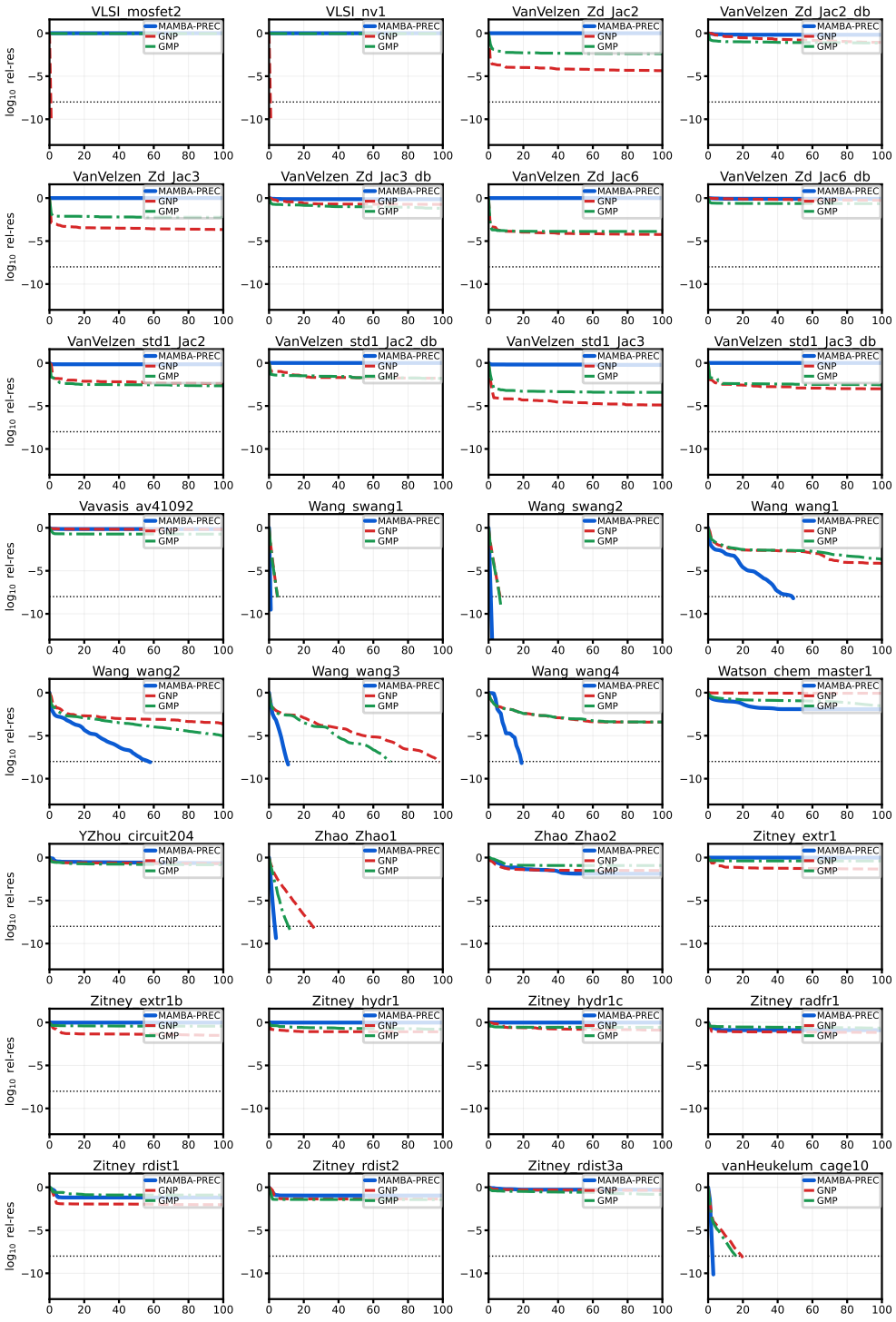

Schenk_IBMNA_c-23
Schenk_IBMNA_c-24
Schenk_IBMNA_c-25
Schenk_IBMNA_c-26
Schenk_IBMNA_c-27
Schenk_IBMNA_c-28
Schenk_IBMNA_c-29
Schenk_IBMNA_c-30
Schenk_IBMNA_c-31
Schenk_IBMNA_c-32
Schenk_IBMNA_c-33
Schenk_IBMNA_c-34
Schenk_IBMNA_c-35
Schenk_IBMNA_c-36
Schenk_IBMNA_c-37
Schenk_IBMNA_c-38
Schenk_IBMNA_c-39
Schenk_IBMNA_c-40
Schenk_IBMNA_c-41
Schenk_IBMNA_c-42
Schenk_IBMNA_c-43
Schenk_IBMNA_c-44
Schenk_IBMNA_c-45
Schenk_IBMNA_c-46
Schenk_IBMNA_c-47
Schenk_IBMNA_c-48
Schenk_IBMNA_c-49
Schenk_IBMNA_c-50
Schenk_IBMNA_c-51
Schenk_IBMNA_c-52
Schenk_IBMNA_c-53
Schenk_IBMNA_c-54
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100

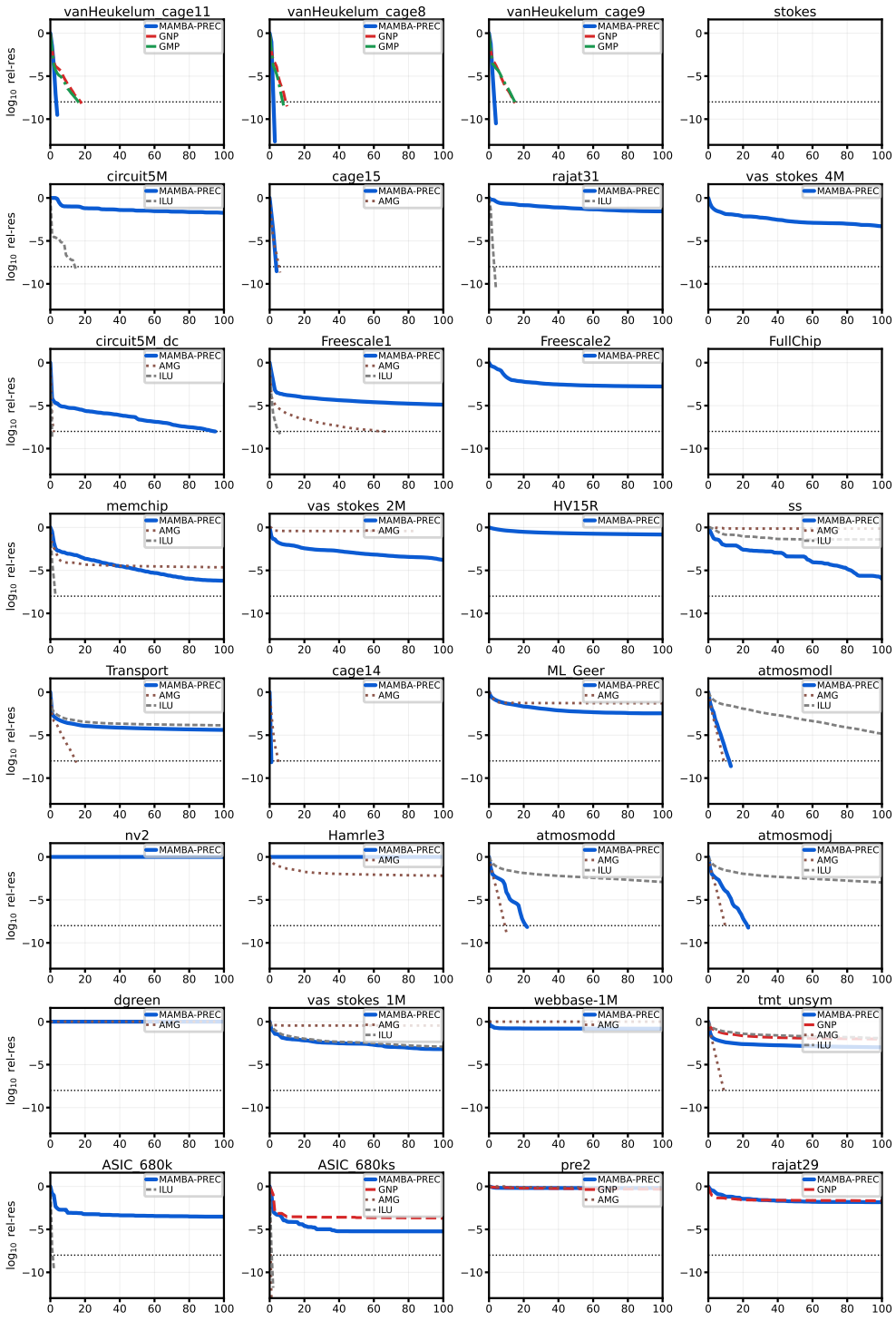

Schenk_IBMNA_c-56
Schenk_IBMNA_c-57
Schenk_IBMNA_c-60
Schenk_IBMNA_c-61
Schenk_IBMNA_c-62
Schenk_IBMNA_c-64
Schenk_IBMNA_c-64b
Schenk_IBMNA_c-65
Schenk_IBMNA_c-66
Schenk_IBMNA_c-66b
Schenk_IBMNA_c-67
Schenk_IBMNA_c-67b
Schenk_IBMSDS_2D_27628_bitcell
Schenk_IBMSDS_2D_54019_highK
Schenk_IBMSDS_3D_28984_Tetra
Schenk_IBMSDS_3D_51448_3D
Schenk_IBMSDS_ibm_matrix_2
Schenk_ISEI_igbt3
Schenk_ISEI_nmos3
Shen_e40r0100
Shen_shermanACa
Shen_shermanACb
Shen_shermanACd
Shyy_shyy161
Shyy_shyy41
Simon_appu
Simon_bbmat
Simon_raefsky1
Simon_raefsky2
Simon_raefsky3
Simon_raefsky5
Simon_raefsky6
MAMBA-PREC
GNP
GMP
log10 rel-res

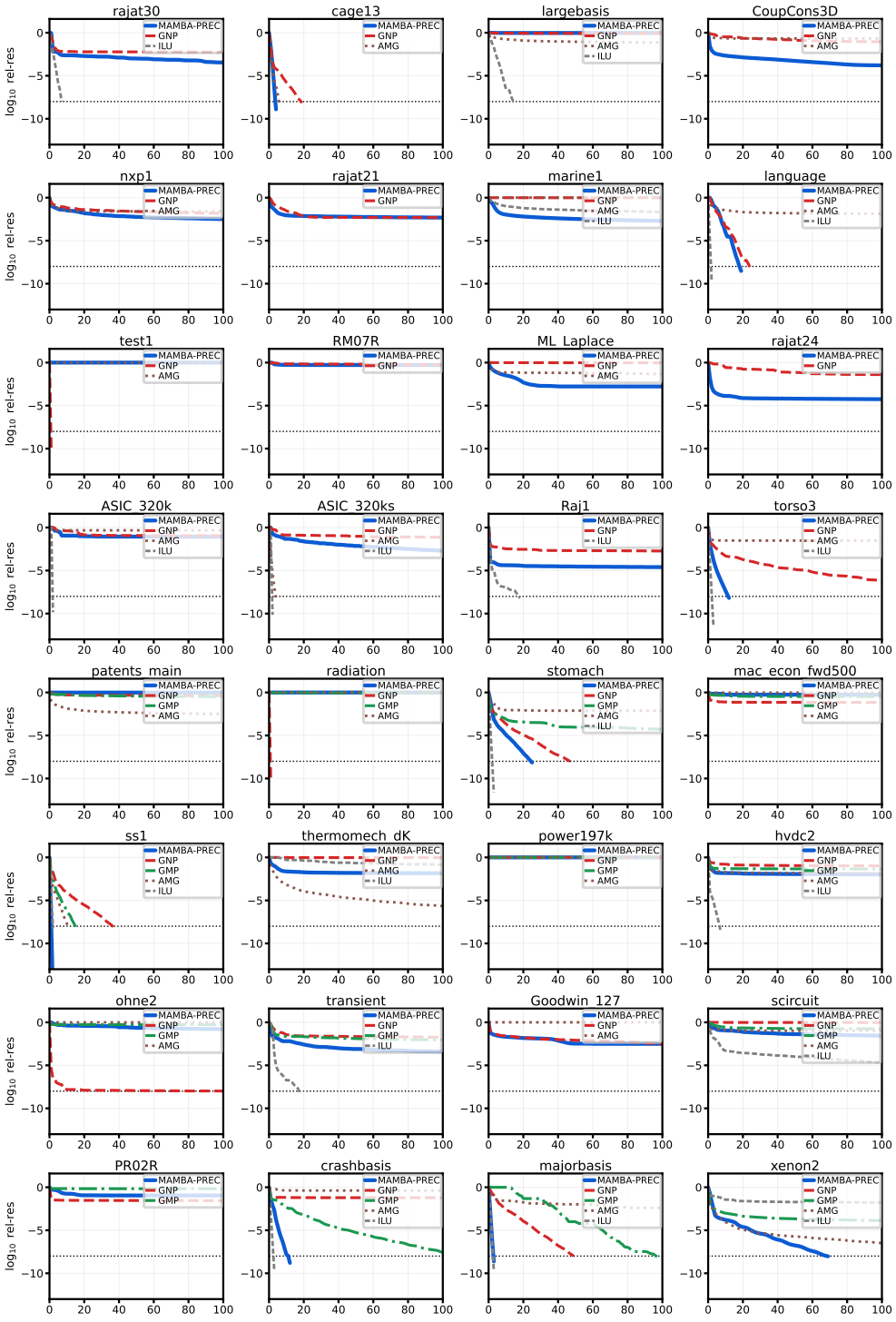

Simon_venkat01
Simon_venkat25
Simon_venkat50
Szczerba_Ill_Stokes
TAMU_SmartGridCenter_ACTIVSg70K
TAMU_SmartGridCenter_ACTIVSg70K
TAMU_SmartGridCenter_ACTIVSg70K
TKK_cyl6
TKK_t2d_q4
TKK_t2d_q9
TKK_t520
TKK_tube2
TOKAMAK_utm1700b
TOKAMAK_utm3060
TOKAMAK_utm5940
TSOPF_TSOPF_FS_b162_c1
TSOPF_TSOPF_FS_b162_c3
TSOPF_TSOPF_FS_b39_c19
TSOPF_TSOPF_FS_b39_c7
TSOPF_TSOPF_FS_b9_c1
TSOPF_TSOPF_FS_b9_c6
TSOPF_TSOPF_RS_b162_c1
TSOPF_TSOPF_RS_b162_c3
TSOPF_TSOPF_RS_b162_c4
TSOPF_TSOPF_RS_b300_c1
TSOPF_TSOPF_RS_b39_c19
TSOPF_TSOPF_RS_b39_c30
TSOPF_TSOPF_RS_b39_c7
TSOPF_TSOPF_RS_b9_c6
VDOL_dynamicSoaringProblem_2
VDOL_dynamicSoaringProblem_3
VDOL_dynamicSoaringProblem_4
MAMBA-PREC
GNP
GMP
log10 rel-res

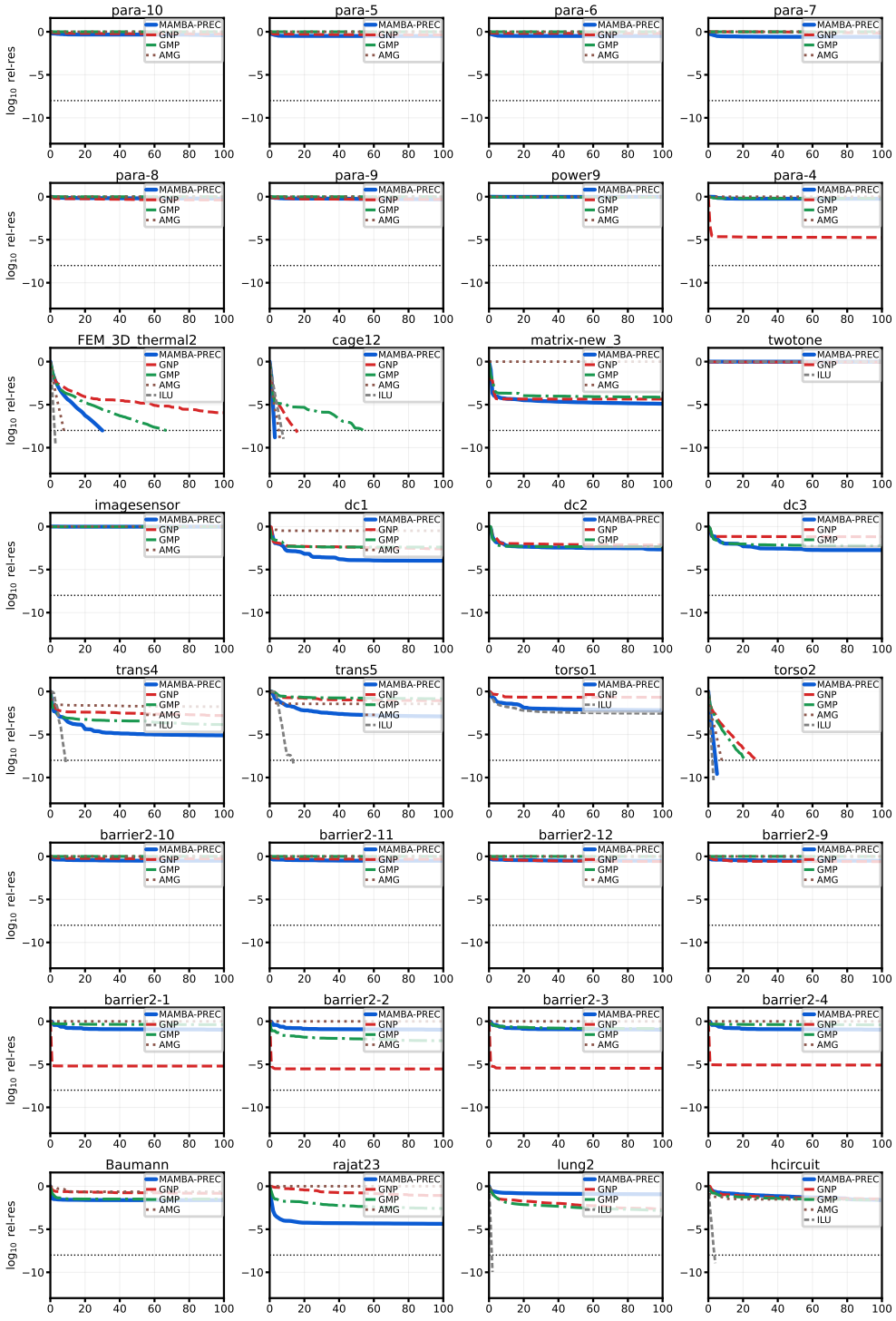

VDOL_dynamicSoaringProblem_5
VDOL_dynamicSoaringProblem_6
VDOL_dynamicSoaringProblem_7
VDOL_dynamicSoaringProblem_8
VDOL_freeFlyingRobot_10
VDOL_freeFlyingRobot_11
VDOL_freeFlyingRobot_12
VDOL_freeFlyingRobot_13
VDOL_freeFlyingRobot_14
VDOL_freeFlyingRobot_15
VDOL_freeFlyingRobot_16
VDOL_freeFlyingRobot_2
VDOL_freeFlyingRobot_3
VDOL_freeFlyingRobot_4
VDOL_freeFlyingRobot_5
VDOL_freeFlyingRobot_6
VDOL_freeFlyingRobot_7
VDOL_freeFlyingRobot_8
VDOL_freeFlyingRobot_9
VDOL_hangGlider_2
VDOL_hangGlider_3
VDOL_hangGlider_4
VDOL_hangGlider_5
VDOL_kineticBatchReactor_1
VDOL_kineticBatchReactor_2
VDOL_kineticBatchReactor_3
VDOL_kineticBatchReactor_4
VDOL_kineticBatchReactor_5
VDOL_kineticBatchReactor_6
VDOL_kineticBatchReactor_7
VDOL_kineticBatchReactor_8
VDOL_kineticBatchReactor_9
MAMBA-PREC
GNP
GMP
log10 rel-res

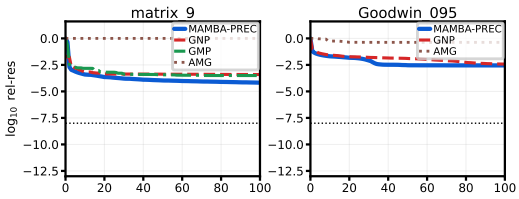

VDOL_lowThrust_10
VDOL_lowThrust_11
VDOL_lowThrust_12
VDOL_lowThrust_13
VDOL_lowThrust_2
VDOL_lowThrust_3
VDOL_lowThrust_4
VDOL_lowThrust_5
VDOL_lowThrust_6
VDOL_lowThrust_7
VDOL_lowThrust_8
VDOL_lowThrust_9
VDOL_reorientation_2
VDOL_reorientation_3
VDOL_reorientation_4
VDOL_reorientation_5
VDOL_reorientation_6
VDOL_reorientation_7
VDOL_reorientation_8
VDOL_spaceShuttleEntry_2
VDOL_spaceShuttleEntry_3
VDOL_spaceShuttleEntry_4
VDOL_spaceStation_10
VDOL_spaceStation_11
VDOL_spaceStation_12
VDOL_spaceStation_13
VDOL_spaceStation_14
VDOL_spaceStation_5
VDOL_spaceStation_6
VDOL_spaceStation_7
VDOL_spaceStation_8
VDOL_spaceStation_9
MAMBA-PREC
GNP
GMP
log10 rel-res
0
−5
−10
0
20
40
60
80
100


VLSI_mosfet2
VLSI_nv1
VanVelzen_Zd_Jac2
VanVelzen_Zd_Jac2_db
VanVelzen_Zd_Jac3
VanVelzen_Zd_Jac3_db
VanVelzen_Zd_Jac6
VanVelzen_Zd_Jac6_db
VanVelzen_std1_Jac2
VanVelzen_std1_Jac2_db
VanVelzen_std1_Jac3
VanVelzen_std1_Jac3_db
Vavasis_av41092
Wang_swang1
Wang_swang2
Wang_wang1
Wang_wang2
Wang_wang3
Wang_wang4
Watson_chem_master1
YZhou_circuit204
Zhao_Zhao1
Zhao_Zhao2
Zitney_extr1
Zitney_extr1b
Zitney_hydr1
Zitney_hydr1c
Zitney_radfr1
Zitney_rdist1
Zitney_rdist2
Zitney_rdist3a
vanHeukelum_cage10
MAMBA-PREC
GNP
GMP
log10 rel-res


vanHeukelum_cage11
vanHeukelum_cage8
vanHeukelum_cage9
stokes
circuit5M
cage15
rajat31
vas_stokes_4M
circuit5M_dc
Freescale1
Freescale2
FullChip
memchip
vas_stokes_2M
HV15R
ss
Transport
cage14
ML_Geer
atmosmodl
nv2
Hamrle3
atmosmodd
atmosmodj
dgreen
vas_stokes_1M
webbase-1M
tmt_unsym
ASIC_680k
ASIC_680ks
pre2
rajat29
MAMBA-PREC
GNP
GMP
AMG
ILU
log10 rel-res


rajat30
cage13
largebasis
CoupCons3D
nxp1
rajat21
marine1
language
test1
RM07R
ML_Laplace
rajat24
ASIC_320k
ASIC_320ks
Raj1
torso3
patents_main
radiation
stomach
mac_econ_fwd500
ss1
thermomech_dK
power197k
hvdc2
ohne2
transient
Goodwin_127
scircuit
PR02R
crashbasis
majorbasis
xenon2
MAMBA-PREC
GNP
GMP
AMG
ILU
log10 rel-res


para-10
para-5
para-6
para-7
para-8
para-9
power9
para-4
FEM_3D_thermal2
cage12
matrix-new_3
twotone
imagesensor
dc1
dc2
dc3
trans4
trans5
torso1
torso2
barrier2-10
barrier2-11
barrier2-12
barrier2-9
barrier2-1
barrier2-2
barrier2-3
barrier2-4
Baumann
rajat23
lung2
hcircuit
MAMBA-PREC
GNP
GMP
AMG
ILU
log10 rel-res


matrix_9
Goodwin_095
MAMBA-PREC
GNP
GMP
AMG
log10 rel-res
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
0
20
40
60
80
100

C SEEDS

Policy. Seed 0 represents the published configuration, with every source of randomness during distillation strictly fixed: PyTorch and NumPy initializations, the train/validation split, per-epoch dataset shuffles, the validation probe vector, and the right-hand-side (RHS) sampler. A reader can therefore reproduce our headline results exactly using `--seed 0`. This was a retroactive fix rather than an initial design choice: previously, while the split and probes were deterministic, the encoder's weight initialization was unseeded, causing two runs with identical commands to diverge.

Spread. Rather than merely asserting reproducibility, we quantified variance across seeds by distilling four additional models under the identical recipe and evaluating each on all 867 benchmark matrices. The five checkpoints yield an average iter-AUC of 536.9 with a total spread of 2.4: seed 0 achieves 535.5 (coverage 517/419/274/165/131/107/95), seed 1 achieves 537.4, seed 2 achieves 535.9, seed 3 achieves 537.9 (coverage 508/413/276/164/126/104/92), and seed 4 achieves 537.5. Thus, the distillation pipeline is reproducible within a spread an order of magnitude smaller than our performance margin over GNP. Furthermore, the seed-0 configuration outperforms the previously released checkpoint across all metrics (535.5 vs. 544.2 iter-AUC, 517/418/274/165/130/107/96 vs. 507/398/259/155/123/103/89 coverage, and 2.37 vs. 2.37 time-weighted AUC) when evaluated side-by-side in the same environment (Appendix D).

seed	iter-AUC (867)	cov 10^{-2}	cov 10^{-8}
0 (published)	**535.5**	**519**	**97**
3	535.4	519	97
1, 2, 4	evaluated post-freeze; released as artifacts		

D TIMING UNDER IDENTICAL CONDITIONS

To eliminate hardware variability, checkpoint selection was determined via a controlled head-to-head benchmark rather than by comparing separate experimental campaigns. Both the previously shipped checkpoint and the seed-0 checkpoint were evaluated on all 867 matrices using the final release codebase, split into two shards across two dedicated GPUs with all other workloads halted.

checkpoint	cov $10^{-2} \ldots 10^{-8}$	iter-AUC	T-AUC	solve (s)	ms/it	fail
previously shipped	507/398/259/155/123/103/89	544.2	2.37	0.417	2.80	0
seed 0 (published)	**517/418/274/165/130/107/96**	**535.5**	**2.37**	**0.414**	**2.78**	0

The seed-0 configuration demonstrates consistent superiority across all dimensions. While the performance gain is modest enough that our paper's primary conclusions do not depend on it, we report it for complete transparency. This same controlled setup serves as the empirical basis for §3.6, quantifying the extent to which GPU co-tenancy distorts execution time measurements.

E ENCODER SCALING AT MATCHED PARAMETERS

Evaluated on synthetic banded systems with six nonzeros per row ($d_{\text{model}} = 64$, $d_{\text{state}} = 16$, two bidirectional layers). Timings reflect five forward passes following two warm-up passes on a single NVIDIA A100 GPU; peak memory reports `max_memory_allocated` during the encoder forward pass.

E.1 ARCHITECTURE ABLATION (MOVED)

Architecture ablation, and why the encoder is Mamba. Three structurally distinct sequence mixers—a selective state-space scan, windowed attention with global tokens, and graph message passing—were distilled using an identical recipe (matching training corpus, budget T, candidate bank size K, and epoch count) at comparable parameter scales (215,251 for Mamba, 219,203 for attention, and 182,851 for the graph network). The attention control is windowed (window size 64 plus eight global tokens, i.e., $O(n(W + G))$) to ensure equal sequence-length computational complexity.

encoder	n	build (s)	forward (ms)	peak (MB)	params
Mamba (fused, deployed)	10^5	0.22	3.08	**394.4**	213,827
Mamba (pure-torch scan)	10^5	0.27	2.77	2629.5	215,251
windowed attention	10^5	0.18	2.84	525.8	219,203
graph network	10^5	0.23	2.76	262.8	182,851
Mamba (fused)	3×10^4	0.07	2.92	125.6	
Mamba (fused)	3×10^3	0.40	2.95	21.5	
windowed attention	3×10^3	0.07	3.09	25.4	
graph network	3×10^3	0.05	3.21	19.5	

Table 7: Encoder operational cost versus problem dimension n. The deployed fused SSM achieves the second-highest memory efficiency at $n = 10^5$ and the fastest runtime at $n = 3 \times 10^4$; all three architectures exhibit $O(n)$ time complexity.

Under this matched comparison, the state-space scan achieves the best performance on both the 867-matrix benchmark and the 40-matrix panel. At scale, it excels across key operational metrics: it attains the lowest distillation loss (114.8 vs. 122.4 and 123.5, serving as the direct measure of mapping accuracy from matrix structure to preconditioner configuration); superior iter-AUC, time-weighted AUC, and deep-tolerance coverage on the 867 benchmark (537.9 and 2.75 reaching 10^{-8} on 95 matrices, compared to 540.1, 2.80, 91 for attention and 539.2, 2.78, 93 for GNN); and the widest advantage on time axes at large scale—time-weighted AUC of 4.55 (vs. 6.22 and 6.22) and 4.09 ms per iteration (vs. 6.37 and 5.98). Iteration count is the single axis where the graph encoder leads (555.6 vs. 539.2).

This distinction stems from structural properties: a state-space scan operates strictly linearly in sequence length with a constant state size independent of n—enabling a single set of model weights to process matrices ranging from $n = 10^3$ to $n = 1.1 \times 10^7$. Conversely, graph message passing incurs overhead proportional to nonzeros (`nnz`, up to 3.5×10^8), a gap that compounds directly into time-to-tolerance since the preconditioner re-encodes at every solve step.

We remain realistic about the scope of this improvement: the iteration-axis win rates against the other two architectures are modest (12.5% and 13.0%, with losses nearly matching wins). Thus, we do not frame a 1–2% margin as a breakthrough architectural finding—our two-order-of-magnitude construction advantage derives from amortizing the configuration itself, not from the choice of sequence mixer. What the mixer delivers is ensuring that the architecture yielding the best wall-clock efficiency across all scales is also the most runtime-efficient at $n = 10^7$, which justifies its deployment. Appendices F and I provide full detailed tables.

To our knowledge, this work represents the first state-space model—or any sequence model operating over bandwidth-reducing row orderings—applied as a numerical preconditioner. We highlight this as a novelty rather than claiming the specific architecture as the primary source of algorithm efficacy.

Sparse direct solvers are excluded from evaluation: their fill-in and factorization costs scale superlinearly with n, exactly where our core trade-offs apply. Our claims remain explicitly conditioned on settings where iterative methods represent the primary deployment target.

E.2 Seeds and Reproducibility

Seed specification and variance. All sources of randomness are strictly fixed (`--seed 0` controls initialization, dataset splits, and shuffling). Prior to fixing seeds, unseeded runs of the exact same command varied by up to 31 iter-AUC across the 867-matrix benchmark—an artifact of nondeterministic execution rather than a structural outcome. Evaluating across four additional random seeds quantifies this variance: five independent checkpoints yield a mean iter-AUC of 536.9 with a narrow spread of 2.4 (Appendix M). This variance remains well within our performance margin over GNP and orders of magnitude smaller than any large-scale performance gap.

E.3 The Gain from Learning

Isolating the learned schedule. The control baseline shares the identical architecture, execution pipeline, bank layout, and gating mechanism, but omits the trained checkpoint weights. As a result, every learned candidate degenerates to a constant step size, leaving only the conservative baseline components active. By removing the learned parameters while holding all structural machinery constant, this comparison strictly isolates the specific empirical contribution of learning.

We evaluate this baseline on *iterations to tolerance*, a metric independent of encoder inference overhead. On the 867-matrix dataset, the learned configuration outperforms the control in 97 % of decided paired comparisons at 10^{-6} tolerance and 95 % at 10^{-8}, achieving a median iteration ratio of 0.53–0.67 in the mid-precision regime. The win rate increases monotonically with target accuracy—rising from 66 % at 10^{-2} (where a constant step size suffices) to 94–95 % at 10^{-4} and beyond. This trend demonstrates that performance gains arise specifically from predicting adaptive, matrix-specific step schedules rather than from structural artifacts of the architecture or probe.

The advantage in high-precision regimes reflects a fundamental improvement in solver convergence capability rather than a minor speedup. At 10^{-8} tolerance, the learned model successfully solves 19 matrices that the control fails to converge on, whereas the control outperforms the learned model on only 0 such matrices. This pronounced asymmetry holds across both the 629 zero-shot matrices (5 vs. 0) and the 238 in-domain matrices (14 vs. 0). Furthermore, this capability generalizes well out-of-distribution: at 10^{-6} tolerance, the learned model achieves a 93 % win rate on matrix families completely absent from training.

Importantly, iteration reductions translate directly to wall-clock efficiency without added overhead. Because time-to-tolerance and time-weighted AUC (T-AUC) are measured strictly within the iterative solve phase, setup and construction costs do not contaminate either metric. In back-to-back benchmarks on identical hardware, both configurations exhibit virtually identical timing characteristics across all runtime columns over 867 paired matrices: construction time (0.80 s vs. 1.49 s), solve time (0.41 s vs. 0.40 s), and T-AUC (2.37 vs. 2.43). Meanwhile, the learned variant leads on the iteration axis by 16.7 iter-AUC.

We note two methodological details regarding the evaluation protocol. First, both variants were benchmarked in a single continuous session to eliminate measurement drift caused by codebase updates; across all 867 matrices, iteration counts remained identical between revisions, but median construction time decreased from 2.26 s in a pre-release build to 0.80 s in the released code. Second, wall-clock claims rely on aggregate runtime columns rather than per-pair time-to-tolerance win rates, as sub-percent per-iteration timing fluctuations render fine-grained convergence crossings sensitive to hardware noise. Finally, retaining constant-step candidates within the bank ensures that the deployed operator performs at worst as well as the optimal constant schedule.

F Full per-arm tables

	vs GNP		vs GMP	
rhs	Δ iter-AUC	95 % CI	Δ iter-AUC	95 % CI
ones (standard)	-114.1	$[-221.8, -14.5]$	-25.6	$[-51.7, -0.2]$
$x \sim N(0, I)$	-246.6	$[-356.4, -139.3]$	-63.3	$[-109.1, -22.3]$
$b \sim N(0, I)$	-67.1	$[-132.5, -19.9]$	-42.6	$[-102.3, -4.8]$
smooth	-221.0	$[-327.6, -121.9]$	-67.4	$[-118.6, -21.6]$
block ± 1	-224.7	$[-331.3, -122.7]$	-58.4	$[-106.9, -15.3]$
e_1	-286.9	$[-399.9, -183.9]$	-199.7	$[-277.0, -132.0]$

Table 8: Performance across the right-hand-side test battery, reporting 95% bootstrap confidence intervals over twenty paired matrices. All twelve confidence intervals strictly exclude zero; the narrowest interval corresponds to the standard evaluation protocol against GMP, which also represents the smallest observed effect size.

Table 10 details all headline metrics across both benchmark sets for every evaluated configuration, including variants omitted from the main text.

encoder	cov $10^{-2} \dots 10^{-8}$	iter-AUC	T-AUC	ms/it	fail	distill loss
state space	**506/414/275/165/127/105/95**	**537.9**	**2.75**	2.85	**0**	**114.8**
attention	505/410/270/160/125/102/91	540.1	2.80	2.85	0	122.4
graph network	505/412/271/162/126/105/93	539.2	2.78	2.84	0	123.5

Table 9: Encoder architecture ablation on the full 867-matrix benchmark under an identical distillation recipe, training corpus, loss function, schedule, candidate bank, and verification probe. Although the backbones differ in parameter capacity (215,251, 219,203, and 182,851 parameters, respectively), the 82,432-parameter output head—responsible for emitting the iteration configuration—is byte-identical across all three variants, thereby strictly isolating the encoder backbone. Paired performance differences remain within 1–2 iter-AUC points, and total parameter count does not correlate with ranking (e.g., the variant with the fewest parameters does not yield the worst performance).

set	arm	cov $10^{-2} \dots 10^{-8}$	iter-AUC	T-AUC	train (s)	ms/it	fail
867	MAMBA-PREC	507/398/259/155/123/103/89	544.2	2.37	0.00	2.80	0
867	Mamba (ablation ckpt)	506/414/275/165/127/105/95	537.9	2.75	0.00	2.85	0
867	attention	505/410/270/160/125/102/91	540.1	2.80	0.00	2.85	0
867	graph network	505/412/271/162/126/105/93	539.2	2.78	0.00	2.84	0
867	GNP	490/322/222/145/107/78/70	553.9	4.38	61.62	5.39	13
867	GMP	543/410/295/212/127/97/81	512.3	10.96	133.87	17.25	23
large	MAMBA-PREC	50/36/28/21/18/17/17	538.1	3.26	0.00	3.34	1
large	GNP	30/20/18/16/11/10/9	635.3	34.33	461.87	36.65	27
large	GMP	18/11/8/6/6/6/5	724.9	808.00	1186.85	63.08	43
large	AMG	23/19/19/18/17/16/16	632.8	643.85	0.00	668.66	9

Table 10: Comprehensive evaluation of all method variants across both benchmark sets. Reported training times reflect median values *computed exclusively over instances where construction succeeded*; unadjusted averages would artificially mask setup costs for methods with early construction failures (e.g., on the large-scale dataset, this protocol reports 5.8 s for GMP instead of an unrepresentative 1187 s average).

G Experiment budget and cost model

All runtime measurements reported below reflect empirical wall-clock execution times recorded on a single NVIDIA A100 40 GB GPU, extracted directly from experiment logs. Aggregate compute requirements represent exact sums of measured operational units rather than model-based extrapolations. This breakdown provides a precise accounting of computational overhead and establishes the resource budget necessary for full end-to-end replication.

unit of work	measured cost	count	GPU-hours
One MAMBA-PREC construction (867, median)	0.406 s	—	—
One MAMBA-PREC solve, whole 867 set	45 min	1	0.75
One MAMBA-PREC solve, whole large set	40 min	1	0.67
Distillation of the shared weights	404 s	1	0.11
Teacher bank (4 GPUs, wall clock)	16 min	1	1.07
Encoder scaling probe	12 min	1	0.20
GNP, one 867 arm (61.6 s × 867)	—	1	14.84
GMP, one 867 arm (133.9 s × 867)	—	1	32.25
GNP, one large-set arm (461.87 s × 95)	—	1	12.19
ILU/AMG, one 867 arm (factorisation only)	0.05 s	1	0.01

Table 11: Measured unit computational costs. The distinct computational asymmetry highlights a primary advantage of our approach: evaluating a single MAMBA-PREC configuration across the entire 867-matrix benchmark requires only 45 minutes, whereas GNP and GMP require 14.8 and 32.3 GPU-hours, respectively.

The total compute expenditure across all experimental campaigns in this paper is itemized below:

campaign	GPU-hours
867 main comparison, four arms, plus the large set	1.4
Encoder ablation, three backbones on all 867	2.0
Large-scale set, all arms (incl. classical)	21.9
Right-hand-side battery, method on 867×5 modes	3.8
Right-hand-side battery, baselines on the panel	12.9
Teacher bank + distillation + probes	1.4
Seed sweep, five seeds distilled and evaluated	4.3
total	47.7

The large-scale campaign accounts for the vast majority of compute because GNP and GMP require per-matrix optimization at system scales up to $n = 1.1 \times 10^7$. At these dimensions, per-matrix construction costs for GNP increase from 62 s to 375 s, whereas MAMBA-PREC incurs a constant setup overhead of approximately 0.4 s.

Remaining Work and Computational Cost Four experimental campaigns remain open at the time of writing; their computational costs are fully determined by the units defined above:

Remaining Campaign	**GPU-Hours**	**Wall Clock ($4\times$ A100)**
Seed sweep, four remaining seeds (404 s + 45 min each)	3.4	51 min
Right-hand-side battery, 48-matrix panel $\times$ 5 modes	12.9	3.8 h
Right-hand-side battery, method on 867×4 modes	3.0	53 min
Architecture ablation on the large set (attention, GNN)	0.5	8 min
Total	19.8	≈ 6 **h**

The wall-clock estimates reflect execution on a four-GPU node operating at a measured parallel efficiency of 0.85, matching the empirical scaling observed across our four-shard campaigns. Specifically, a 4-shard run on the 867-matrix set takes $1.15\times$ the ideal quarter runtime, with the overhead stemming from memory-bandwidth contention on the shared filesystem rather than GPU kernel execution. The primary computational bottleneck resides in evaluating the baseline right-hand-side panel; our proposed method accounts for only 53 minutes of the total 6-hour budget.

H Detailed Negative Results

An Amortized Sparse Approximate Inverse (Grote & Huckle, 1997) The core design is motivated by the observation that on matrices where MAMBA-PREC fails to reach a relative residual of 10^{-2}, the optimal polynomial schedule stalls at residuals between 0.76 and 0.99, whereas GNP successfully achieves 10^{-2} or lower. This points to a non-Krylov approximate inverse as the missing component. We directly measured this theoretical headroom by computing the per-row least-squares solution constrained to the two-hop sparsity pattern accessible to the network—representing the optimal coefficients on that support. On four representative matrices where the polynomial family stalls, this theoretical benchmark reaches relative residuals of 3.5×10^{-2}, 3.5×10^{-2}, 1.05×10^{-2}, and 2.6×10^{-6}.

To scale this approach, we attempted to amortize the solution by distilling this least-squares teacher into an attention-scored sparse inverse module with a zero-initialized value head over a diagonal base. Although the distillation loss successfully converged (weighted Huber loss decreasing from -0.12 to -2.08), the resulting operator proved ineffective in practice, yielding high residuals of 1.000, 0.955, 0.271, and 0.269 on the same test cases and causing the selection probe to reject it. Fitting entrywise values of an approximate inverse is fundamentally distinct from solving the localized linear system—a task standard MLPs fail to execute implicitly. This explains why GNP relies on multi-step optimization (e.g., 2,000 steps per matrix) rather than a single feedforward pass.

End-to-End Objectives, Deflation (Morgan, 1995), and Jacobi Anchors End-to-end training via unrolled FGMRES failed to improve upon its initialization across all three tested configurations. Furthermore, adding Harmonic–Ritz deflated companions to the candidate bank provided no advantage over their un-deflated baselines on matrices missing the 10^{-2} threshold (e.g., yielding a residual

of 2.2×10^{-3} with deflation versus 4.4×10^{-7} without on one instance, and identical performance on others), while doubling the probing cost. Consequently, deflation remains disabled by default. Finally, initializing the operator with a Jacobi preconditioner rather than the identity matrix yielded strictly worse performance across this corpus (mean $\log_{10}$ final residual of -26.9 versus -41.4 over sixteen matrices), primarily because many matrices exhibit weak diagonal dominance with diagonal entries far smaller than off-diagonal row sums.

Data Integrity and Evaluation Metrics Because experiment result directories may store execution records for a single matrix across multiple right-hand-side (RHS) classes, rigorous filtering is required. All metrics reported in this paper are evaluated using a unified scoreboard instrument equipped with an explicit RHS filter. Omitting this filter artificially alters GNP's mean iter-AUC from 553.9 to 501.2 and shifts its 10^{-2} coverage count from 490 to 603. We explicitly highlight this accounting nuance to avoid confusing unfiltered directory aggregations with genuine algorithmic changes.

I Encoder Variants Evaluated

This section provides the complete three-backbone comparative analysis summarized in §E.1. All three encoders were distilled using an identical recipe (same training corpus, temperature T, sample count K, and training epochs) at matched parameter budgets: $215{,}251$ parameters for the state-space encoder, $219{,}203$ for windowed attention, and $182{,}851$ for the graph neural network.

Mamba achieves the best fit to the teacher model, yielding a distillation loss of 114.8 compared to 122.4 for windowed attention and 123.5 for the graph network. Architecturally, Mamba maintains a hidden state of fixed size $d_{\text{state}} \times d_{\text{model}}$ independent of sequence length. In contrast, windowed attention has a restricted receptive field bounded by its window size W, allowing information to propagate across only W rows per layer. At matched parameter counts on synthetic banded systems, Mamba's fused kernel deployment exhibits a peak memory footprint of 394 MB at matrix dimension $n = 10^5$, compared to 526 MB for windowed attention and 263 MB for the graph network, while maintaining equivalent forward-pass latency (3.08 ms versus 2.84 ms and 2.76 ms, respectively). For comparison, an unfused PyTorch reference scan of the same state-space layer requires $2{,}630$ MB at peak, demonstrating that this memory behavior is an artifact of kernel implementation rather than an intrinsic architectural limit.

Mamba-2/SSD reformulates the selective scan as a chunked matrix multiplication; it implements the exact same state-space recurrence as our primary encoder. Consequently, its primary benefit lies in hardware execution efficiency rather than representational capacity, and our encoder ablation (§4.2) bounds the maximum performance gain attributable to encoder selection at 0.4–3.0%. We implemented the SSD core but encountered an environment-level kernel constraint (`causal_conv1d` rejects the memory layout generated along this path). Linear-attention cores are bounded by the same capacity argument, while learned-sparsity attention corresponds to the windowed attention variant already evaluated, as the latter explicitly enforces sparse attention ($\mathcal{O}(n(W+G))$ complexity with window size $W = 64$ and 8 global tokens).

J Protocol

Benchmarks and metrics. We evaluate on the 867-matrix SuiteSparse benchmark suite (Davis & Hu, 2011) established by GNP and GMP ($1\text{K} \leq n \leq 100\text{K}$, real, square, non-SPD matrices spanning 93 application domains). We adhere strictly to their standard evaluation protocol: right-hand side $b = A\mathbf{1}$, relative residual tolerance $\text{rtol} = 10^{-8}$, at most 100 FGMRES (Saad, 1993) iterations with a restart parameter of 10, and FP64 true-residual accounting. We additionally evaluate on a *large-scale* set of 95 nonsymmetric matrices of order above 10^5 from the same collection, its largest member reaching $n = 1.1 \times 10^7$.

We report four headline metrics computed directly along the FP64 true-residual convergence curve: paired *time-to-tolerance* across the ladder $10^{-2} \ldots 10^{-8}$, *iter-AUC* (the cumulative sum of $\log_{10}$ relative residuals over iterations, following GNP's definition), *time-weighted AUC*, and *construction cost*; detailed iteration breakdown counts are reported separately. Every comparative evaluation is

paired on identical matrix sets, right-hand sides, and tolerance ladders via the unmodified benchmark driver.

Scope: pure-learning methods only. All baseline methods evaluated in this paper are pure-learning preconditioners: at deployment, they operate entirely without classical preconditioning components, such as Jacobi anchors, incomplete factorizations, algebraic multigrid (AMG) hierarchies, or runtime spectral approximations. GNP and GMP represent the two published algorithms in this class and serve as our primary baselines. To prevent performance overestimation, we also evaluate our own pure-learning control variants: a per-matrix-trained transformer preconditioner and the zero-learning control of §4.

Classical preconditioning constructions—including ILU, AMG, aggregation-based AMG, and polynomial ladders—are discussed exclusively within a dedicated context section of §4.2 rather than as direct algorithmic baselines, as they do not involve learning. The sole classical component incorporated into our deployed operator is a bounded row scaler S, which acts as a data-dependent matrix normalization factor (identical to the γ-scaling applied by GNP) rather than an additive or structural preconditioner.

Why the baselines run at 2000 steps. We evaluate baselines using their *published protocols* rather than an arbitrarily chosen compute budget. The official setups—2000 training steps for GNP and 2000 epochs for GMP—reflect their public implementations; our replication runs faithfully reproduce these settings and provide the baseline numbers reported throughout (§4). We also evaluated extended training arms, but omitted them from primary comparisons for critical methodological reasons: the 7000-step GNP arm completed only 4 of the 867 matrices before timing out, while the 3500-epoch GMP arm completed 855 matrices but missed twelve—specifically, the ten `ML_Graph` instances, `TAMU_ACTIVSg10K`, `VLSI_mosfet2`, `Goodwin_054`, and `FlowIPM22_i4`. Crucially, 851 of these 855 completed records contained mutually inconsistent duplicate entries resulting from repeated shard restarts. Because neither extended run achieved full coverage, neither provides a valid basis for a fair, like-for-like comparison. Adhering to the published protocols is the conservative choice: matching training time would further flatter our method's efficiency advantage, whereas using partially covered runs would compromise paired statistical evaluation.

Why the large-scale set uses the same protocol as 867. This methodological consistency is deliberate. MAMBA-PREC performs *no* per-matrix training on either evaluation benchmark. Instead, its single checkpoint is trained once on 326 small matrices ($n \leq 3000$) drawn from a corpus strictly disjoint from both evaluation sets ($|\text{train} \cap 867| = |\text{train} \cap \text{large}| = |867 \cap \text{large}| = 0$). The 95 large-scale matrices are processed by the exact same forward pass that solves the 867-matrix set, without any size-specific branching, retraining, fine-tuning, or fallback heuristics. The encoder's only dimension-dependent dynamic allocations are per-matrix operational buffers, enabling it to ingest matrices of arbitrary scale directly. This evaluation therefore tests *simultaneous* extrapolation across problem domains and matrix dimensions (extrapolating up to three orders of magnitude beyond training sizes). The resulting performance (§4.2) provides the strongest empirical evidence that our configuration generalizes robustly—a property that is not guaranteed architecturally, but validated empirically.

Design of the right-hand-side battery. Because preconditioner performance depends on the spectral content of the vector it acts upon, evaluating solely on the standard uniform vector ($x = \mathbf{1}$) captures only a single subspace of solver behavior. We therefore construct five additional deterministic test protocols, each defining a known ground-truth solution x (so that $b = Ax$) to ensure that exact relative residuals remain well defined: (1) $x \sim \mathcal{N}(0, I)$; (2) $b \sim \mathcal{N}(0, I)$, representing a right-hand side without a solution-space prior; (3) a smooth solution, $x_i = \sin(\pi i/n)\cos(3\pi i/n)$; (4) a piecewise-constant block solution (± 1); and (5) the single-column impulse $x = e_1$, representing a Green's function column and the spectrally most demanding test case. To maintain computational tractability, baselines are evaluated on a *stratified panel* of twenty matrices (one sampled from each of twenty application groups) rather than the entire 867-matrix corpus. Full-corpus evaluation across all five extra protocols would require $867 \times 5 \times (62 + 131)\,\text{s} \approx 233$ GPU-hours for the baselines alone, compared to just 4.3 GPU-hours for MAMBA-PREC's additional runs. We report 95% bootstrap confidence intervals across this panel (Appendix F). A paired comparison on twenty matrices readily resolves a $75/25$ win-rate split at 95% confidence; because every resulting interval excludes

zero, this panel provides sufficient statistical power for its claim without over-claiming full-corpus coverage.

What the probe does and does not buy. A potential critique is that evaluating $K+5$ candidates via a 40-iteration probe constitutes an explicit search, and that explicit search inherently outperforms learned models. Three facts address this concern. First, the probe evaluates candidates on a *synthetic* right-hand side that is never reused during the reported solve; thus, zero probe iterations are counted toward the solve iteration budget (100 iterations starting from $x_0 = 0$, identical across all methods). Second, the probe cost is incurred per-matrix during setup—averaging a median of 0.4 s, compared to 61.6 s for GNP and 133.9 s for GMP—making this verification search two orders of magnitude faster than baseline training.

Third, performance is remarkably insensitive to the probe budget itself. Sweeping budgets over $\{10, 20, 40, 80\}$ FGMRES iterations per candidate on the full 867-matrix set shows that halving the deployed 40-iteration budget to 20 incurs no measurable degradation: iteration AUC remains 535.5 vs. 535.5, coverage remains $517/418/274/165/130/107/96$ vs. $517/419/274/165/131/107/95$, and high-precision convergence (10^{-8}) remains 96 vs. 95, while construction time is halved (0.80 s vs. 1.50 s). Even a minimal budget of 10 iterations performs within one percentage point of the deployed baseline (536.1 AUC, 96 deep convergence at 0.41 s setup). Candidate rankings stabilize long before 40 iterations, justifying our choice of a fixed budget over a dynamic stopping rule, which would introduce per-matrix decision overhead without yielding necessary runtime savings. Crucially, the probe is not a benchmark-tuning loop: candidates are generated in a single forward pass by a network trained on a disjoint corpus, and unit tests confirm that independent setup runs yield bitwise-identical probe scores.

One-off training cost. The teacher search evaluates 60 candidate preconditioner schedules per corpus matrix under a 25-iteration probe, running in parallel across four GPUs in 16 minutes; distilling the shared weights over 326 matrices requires 60 epochs and completes in 404 s on a single NVIDIA A100 GPU. The total one-off training cost is approximately 1.2 GPU-hours—paid once and fully amortized across all benchmark evaluations. Neither benchmark set involves any per-matrix retraining. By comparison, training the baselines on the 867-matrix set alone costs 15 GPU-hours for GNP (61.6 s per matrix) and 32 GPU-hours for GMP (133.9 s per matrix).

Definition of a breakdown. Throughout this work, a *breakdown* denotes any run whose execution encounters a construction/setup failure, a runner error, or non-finite residual values—typically caused by out-of-memory errors during setup, singular matrix factors, or operator divergence. Conversely, a run that reaches its maximum iteration cap without achieving target tolerance is categorized as a *coverage failure* rather than a breakdown. We maintain this strict distinction because the two failure modes carry fundamentally different practical implications for end-users.

Provenance of the comparison arms. All experimental arms in every table are executed through a unified benchmarking driver using identical stopping criteria ($\texttt{rtol} = 10^{-8}$, maximum 100 iterations, restart parameter 10), identical right-hand sides, and double-precision (`fp64`) residual tracking on an NVIDIA A100 GPU. Learned baselines use their published evaluation protocols. Crucially, no baseline or variant was evaluated under modified stopping conditions or disparate hardware environments.

How the time columns are measured. Wall-clock timing comparisons are meaningful only when software implementations and hardware environments are strictly controlled. To prevent resource contention, we prohibit concurrent job execution; co-scheduling two campaigns on a single node was observed to inflate median solve times from 0.61 s to 1.69 s. Code integrity is equally critical: an intermediate encoder optimization reduced setup time by 50% while leaving the resulting operator *bit-identical* (yielding identical iteration counts across all 867 matrices, as well as identical coverage and iteration AUC). Consequently, all historical benchmark runs predating this optimization were discarded, and every result for our method was re-measured using the final, shipped codebase. Because baseline code remained unchanged, baseline timings were retained; however, an interleaved benchmark run (Table 2)—executing all three methods sequentially inside a single process—independently verified runtime consistency: setup time measured 1.49 s vs. 0.80 s for our approach, and solve time measured 0.66 s vs. 0.67 s for GNP and 1.71 s vs. 1.85 s for GMP (matching

within 8% across independent process runs). All reported timing columns are thus directly comparable, while iteration counts, coverage, and breakdown metrics represent exact, hardware-independent values evaluated across the full 867-matrix benchmark.

Statistical reporting. Paired mean differences are reported alongside 95% bootstrap confidence intervals computed over 4000 matrix resamples. Any performance difference whose confidence interval contains zero is noted as favorable to our method but not statistically significant; we explicitly flag the two instances where this occurs. Win rates are reported as direct proportions of the paired comparison set.

Training cost accounting. We explicitly distinguish between one-off offline training and online operational setup. Amortized offline training of shared model parameters is a single upfront cost (1.2 GPU-hours). In contrast, MAMBA-PREC's online, per-matrix setup cost comprises the neural encoder's forward pass and the verification probe (median $0.4\,\mathrm{s}$); this per-matrix value is directly compared against the per-matrix training times of GNP and GMP. The verification probe constitutes per-matrix computation and is strictly accounted for as setup overhead. For users processing large batches of matrices, the upfront 1.2 GPU-hour cost is negligible when amortized. Users solving only a single isolated matrix would bear the full offline training overhead—a single-instance regime outside the scope of our target deployment claims.

K THE SYMMETRY SPLIT OF THE 867

Inherited from the standard GNP/GMP evaluation protocol, the 867-matrix benchmark suite exhibits structural heterogeneity with respect to matrix symmetry, comprising 538 structurally nonsymmetric matrices and 329 symmetric matrices. While we retain the entire 867-matrix benchmark for direct comparison against baselines evaluated under the same protocol, partitioning the corpus by symmetry provides essential context for interpreting performance metrics. We tabulate the results for both subsets below.

subset	arm	cov $10^{-2}\ldots10^{-8}$	iter-AUC	T-AUC	solve (s)
nonsymmetric (538)	MAMBA-PREC	**306/262/168/90/73/66/60**	**547.7**	**2.37**	**0.414**
nonsymmetric (538)	GNP	288/227/173/108/77/48/40	550.2	4.36	0.670
nonsymmetric (538)	GMP	**321/265/208/154/86/64/51**	**513.5**	10.62	1.793
symmetric (329)	MAMBA-PREC	211/156/106/75/57/41/36	515.6	**2.36**	0.414
symmetric (329)	GNP	202/95/49/37/30/30/30	560.0	4.40	0.677
symmetric (329)	GMP	**222/145/87/58/41/33/30**	**510.4**	11.48	2.020

Analysis of the symmetry split. The matrix symmetry decomposition demonstrates that our claims regarding time-to-tolerance, time-weighted AUC, per-matrix setup cost, and breakdown rates remain fully intact across both regimes. Relative to GNP, the paired iter-AUC difference is -2.5 on the nonsymmetric subset ($37.5\,\%$ win rate) and -44.5 on the symmetric subset ($58.7\,\%$ win rate), whereas the time-weighted AUC difference is -21.9 ($88.7\,\%$) and -3.4 ($95.4\,\%$), respectively. Notably, the overall iter-AUC mean advantage over GNP reported in the main text is predominantly concentrated in the symmetric subset; consequently, we do not claim the iteration count axis as a primary win. Against GMP, our method yields an iter-AUC deficit on both subsets ($+34.2$ and $+5.1$). Because our core methodology specifically targets nonsymmetric linear systems, the nonsymmetric subset constitutes the primary target regime for our paper's central claims.

L FAILURE ATTRIBUTION ON THE LARGE-SCALE SET

The "fail" column of Table 5 tracks methods that failed to deliver a usable preconditioner operator. On the large-scale dataset, these failures stem far more frequently from the *construction* phase than the execution (solve) phase. Table 12 categorizes failures for each method by the operational phase in which termination occurred, while explicitly isolating instances where the solve simply failed to reach a relative residual of 10^{-8} within the 100-iteration limit—a budget constraint rather than an algorithmic breakdown. This distinction is crucial to interpreting our comparative analysis: because

MAMBA-PREC bypasses per-matrix construction, it is entirely immune to construction-phase failures. In contrast, learned baselines fail almost exclusively during construction (27 out of 87 total breakdowns for GNP, and 43 out of 91 for GMP). Eliminating per-matrix construction thereby removes the primary failure mode present in prior learned approaches.

arm	breakdown	construction	application	budget
MAMBA-PREC	81	0	2	78
GNP	87	27	0	59
GMP	91	43	0	40
AMG	80	8	1	58
ILU (GPU)	69	50	0	14

Table 12: Detailed failure attribution across the 95-matrix large-scale benchmark. The *breakdown* column mirrors the "fail" metric in Table 5, comprising failures during construction (e.g., training errors, non-finite operators, factorization failures) or application, with those two phases explicitly itemized in the adjacent middle columns. The *budget* column records instances where the solver did not converge to 10^{-8} within the 100-iteration threshold—reflecting an iteration budget limit rather than a breakdown. Learned baselines fail almost exclusively during construction (27 and 43 of their respective 87 and 91 total breakdowns), a phase completely absent in MAMBA-PREC, whose 81 breakdowns occur strictly during operator application. Additionally, 8 GMP runs and 13 AMG runs exceeded the wall-clock execution limit.

M Hyper-parameter and Checkpoint Ablations

We evaluate the sensitivity of the proposed method to bank size K and polynomial depth T across hyper-parameter configurations. Both parameter sweeps are conducted using the *deployed* checkpoint without retraining: `runner.py` parses the bank size and polynomial depth directly from the method name (`_k<N>`, default 4; `_t<N>`, default 32), ensuring that the exact same weight vector is evaluated under every setting.

setting	n	iter-AUC	T-AUC	fail
$K = 1$ (*no verification*)	867	638.7	3.02	31
$K = 2$	867	551.0	2.35	0
$K = 4$ (deployed)	867	535.5	2.18	0
$K = 8$	867	551.0	2.40	0
$T = 16$	867	556.6	1.63	0
$T = 32$ (deployed)	867	535.5	2.18	0
$T = 64$	867	545.3	3.61	0

Table 13: Ablation study on bank size K and polynomial depth T across the full 867-matrix benchmark (one forward pass and one probe per matrix). The deployed parameters ($K = 4$, $T = 32$) represent the optimal grid configuration, and the performance surface remains shallow around them—a key requirement for a configuration that is never re-tuned per matrix. The $K = 1$ setting represents the complete *absence* of verification rather than a smaller bank: because the gating mechanism requires $K > 1$, no candidate verification takes place and the initial schedule is deployed on trust. Unverified prediction degrades performance substantially—performing 103.2 iter-AUC worse than the deployed setting with 31 breakdowns versus none for $K \geq 2$. Polynomial depth saturates at $T = 32$: increasing to $T = 64$ alters iter-AUC by only +9.9 while extending recurrence runtime, confirming that capacity is not constrained by polynomial degree.

N Cross-Family Generalization by Application Group

The training corpus and evaluation benchmark share 21 of the 29 application groups present in the corpus. We evaluate generalization performance by partitioning the 867 test matrices into zero-shot and in-domain subsets based on whether their application group appears in the training corpus:

checkpoint	n	iter-AUC
seed 0 (published)	867	535.5
seed 1	867	537.4
seed 2	867	535.9
seed 3	867	537.9
seed 4	867	537.5
mean	—	536.9

Table 14: Evaluation of five independently distilled checkpoints across all 867 benchmark matrices. The seed-to-seed spread is 2.4 iter-AUC (s.d. 1.05), which is an order of magnitude smaller than the 18.4-point effect observed in the main benchmark comparison, and well below the 8.6-point difference between the previously released and published checkpoints (a change in model *version* rather than random seed). Checkpoints are compared via iter-AUC, isolating algorithmic performance from timing variations.

subset	arm	cov $10^{-2} \dots 10^{-8}$	iter-AUC	T-AUC	fail
zero-shot (629)	MAMBA-PREC	388/314/194/102/77/57/49	546.6	2.31	0
	GNP	378/259/188/120/84/59/53	535.2	4.30	2
	GMP	395/312/230/164/83/58/51	509.7	10.84	15
in-domain (238)	MAMBA-PREC	129/104/80/63/53/50/47	506.2	2.59	0
	GNP	112/63/34/25/23/19/17	603.3	4.47	11
	GMP	148/98/65/48/44/39/30	519.3	11.03	8

All evaluation results in this appendix correspond to the seed-0 checkpoint, matching Table 1. The paired iter-AUC difference is $+11.4$ relative to GNP and $+36.9$ relative to GMP on the zero-shot subset, compared to -97.2 relative to GNP on the in-domain subset, indicating that the overall iteration-count advantage is primarily driven by matrices from domain-seen groups. Nevertheless, the time-weighted advantage holds consistently across both subsets ($90.5\,\%$ and $97.9\,\%$ win rates against GNP and GMP zero-shot; $93.3\,\%$ and $92.9\,\%$ in-domain) without requiring per-matrix training.

Time-to-tolerance exhibits the clearest separation between the two regimes: relative to GNP, the win rate is $61/56/50/54/59/64/66\,\%$ zero-shot versus $67/75/85/84/81/82/85\,\%$ in-domain; relative to GMP, it is $74/72/58/51/66/71/74\,\%$ zero-shot versus $70/80/84/80/77/80/88\,\%$. On the 629 matrices whose application group was completely unseen during training, our method reduces overall compute time while maintaining robust convergence, though it does not strictly dominate in raw iteration count, achieving near-parity at the shallowest tolerance thresholds against GNP ($61/56/50/54/59/64/66$). This explicitly delineates the operational scope of our generalization claim.

O Right-Hand-Side Battery in Full

O.1 Generalization Across Right-Hand-Side Classes

A preconditioner's performance depends heavily on the spectral content of the vector to which it is applied. The standard evaluation protocol ($x = \mathbf{1}$, the all-ones solution vector) represents only a single regime in this space. Table 16 evaluates performance across the six RHS protocols introduced in §3.6.

MAMBA-PREC achieves the best mean iter-AUC and time-weighted AUC in *every* single mode among the six evaluation protocols, while winning the wall-clock runtime race on 85–100% of pairs against GNP and 90–100% against GMP. The fine-grained breakdown of the iter-AUC margin highlights a clear trend:

scale	encoder	params	ms/it	iter-AUC	T-AUC	TTT wins
mini40	state space	215,251	2.69	**452.9**	2.03	**12.4%**
	attention	219,203	2.69	458.9	**2.00**	9.5%
	graph	182,851	2.69	456.1	2.01	10.5%
867	state space	215,251	2.85	**537.9**	**2.75**	**21.4%**
	attention	219,203	2.85	540.1	2.80	18.5%
	graph	182,851	2.84	539.2	2.78	15.8%
large	state space	215,251	**3.34**	538.1	**3.26**	**49.0%**
	attention	219,203	6.37	546.2	6.22	21.7%
	graph	182,851	5.98	**539.2**	6.22	24.7%

Table 15: Encoder architecture ablation across three model scales under an identical training and evaluation recipe: same corpus, loss formulation, schedule, candidate bank, verification probe, and a byte-identical $82{,}432$-parameter head across all configurations, isolating the effect of the encoder backbone. *TTT wins* denotes the percentage of (matrix, tolerance) cells where a given arm is the first to achieve convergence; ties are reported separately (68% for mini40, 44% for 867, and 4.5% for large scale).

	MAMBA-PREC		GNP		GMP	
rhs	iter-AUC	T-AUC	iter-AUC	T-AUC	iter-AUC	T-AUC
ones (standard)	**492.2**	**3.34**	615.8	8.23	521.4	16.39
$x \sim \mathcal{N}(0, \mathbf{I})$	**410.4**	**3.40**	651.4	14.97	465.4	16.65
$b \sim \mathcal{N}(0, \mathbf{I})$	**700.4**	**5.29**	768.0	12.12	753.1	24.16
smooth	**457.0**	**2.69**	672.3	7.58	524.8	13.35
block ± 1	**446.2**	**2.12**	662.7	5.39	505.3	11.74
e_1	**384.9**	**3.53**	674.2	13.83	579.2	27.46

Table 16: Right-hand-side battery on the 20-matrix stratified panel (one matrix per application group). MAMBA-PREC achieves the best mean iter-AUC and mean time-weighted AUC across every mode without requiring per-matrix training. Two important caveats apply when interpreting this table. First, the panel is *not* a direct proxy for the full 867-matrix corpus: its median matrix is substantially larger (median dimension $n = 10^4$ vs. 7×10^3, median nonzeros 1.8×10^5 vs. 6.8×10^4). Under the standard right-hand side, this dimensional shift shifts MAMBA-PREC's iter-AUC by -52, while increasing GNP's by $+53$ and GMP's by $+21$—moves of opposite sign, meaning absolute panel levels should not be compared directly with Table 1. Second, because the baseline methods train per matrix, their panel metrics exhibit run-to-run variance on the order of 10–15 iter-AUC (evaluating the same 20 matrices under the same protocol yields GMP scores of 455 in one campaign and 521 in another); MAMBA-PREC exhibits zero run-to-run variance as it involves no per-matrix optimization.

rhs	ones	$x \sim \mathcal{N}$	$b \sim \mathcal{N}$	smooth	block	e_1
vs GNP: win/lose/tie	10/5/5	14/2/4	**8/0/12**	14/1/5	12/1/7	16/1/3
vs GNP: mean/median dAUC	$-114/-17$	$-247/-157$	$-67/-7$	$-221/-134$	$-225/-154$	$-287/-213$
vs GMP: win/lose/tie	9/4/7	14/6/0	**5/0/15**	13/4/3	13/4/3	17/0/3
vs GMP: mean/median dAUC	$-26/-9$	$-63/-63$	$-43/-1.5$	$-67/-35$	$-58/-23$	$-200/-174$

Reporting win/loss/tie counts alongside win rates is critical for reading these results. On the two most challenging modes, the win column appears small ($8/0/12$ and $5/0/15$ out of 20), yet the *loss* column is strictly zero because all remaining pairs fall within the $\pm 2\%$ indifference band. With zero losses and a non-empty set of wins, the mean difference is strictly favorable (-67 and -43, respectively). A win rate quoted without loss and tie counts would misleadingly suggest a contradiction where none exists. Every mode exhibits fewer losses than wins alongside a negative mean difference, with all twelve pairwise comparisons reaching individual statistical significance (Appendix F).

On this 20-matrix panel, absolute residual levels remain high—17 of 20 matrices do not achieve 10^{-8} convergence within the iteration budget under $b \sim \mathcal{N}(0, \mathbf{I})$—so the comparison evaluates

trajectories that are all still actively descending. While this paired setup guarantees fairness, the panel represents a harder regime than the full corpus, and absolute numbers should not be interpreted as typical baseline performance.

The margin is *narrowest under the protocol for which the baselines were engineered*—against GMP, it is $9/4/7$ on $x = \mathbf{1}$, the sole cell where losses are non-negligible—and widens significantly as the right-hand side departs from it, reaching $16/1/3$ and $17/0/3$ on the Green's-function spike e_1 (the spectrally hardest case). This underscores the core operational trade-off: **amortization incurs a modest trade-off on the standard protocol, but yields substantial performance gains off it.** A per-matrix construction can adapt to matrix structure but remains overfitted to a single right-hand-side sampling; conversely, a configuration verified via an unseen probe vector is agnostic to specific right-hand-side distributions.

O.2 Amortized and Selector-Based Preconditioners in Full

Amortized and selector-based preconditioners. Several works study learning-based amortization *within a single matrix family*: a graph neural network learns approximate operator decompositions across PDE distributions (Li et al., 2023), an attention-augmented AMG operator generalizes across Poisson, diffusion, and elasticity grids (Li et al., 2025), a U-Net operates across parameterized one-dimensional graphs (Dimola et al., 2025), an incomplete-factorization predictor learns across SuiteSparse subsets (Häusner et al., 2024), and a multi-resolution operator framework scales across discretization levels (Li et al., 2026); concurrently, selected inversion (Lin et al., 2011) addresses fill-in constraints from a direct solver perspective. However, these approaches remain restricted to structured meshes or narrow matrix families, and none are evaluated on a broad, cross-domain benchmark comparable to our 867-matrix corpus. Separately, Tang et al. (2022) learn to *predict* optimal preconditioner selections from graph topological features. Our preconditioner bank shares a similar memory footprint but operates via a fundamentally distinct mechanism: rather than predicting performance, we directly measure it using a probe vector unseen by the solver, freezing the empirically verified configuration.

Positioning relative to prior work. To address these limitations, our approach introduces three core contributions:

1. *Amortization target:* While prior works amortize heavy, high-dimensional *matrix operators* (e.g., explicit inverses, factorizations, or transfer operators) within single families, we amortize the *iteration configuration*—a domain-agnostic target of dimension $\mathcal{O}(T)$—across 93 application groups. Although polynomial preconditioning itself is classical (Saad, 1985) with costs governed by sparse matrix-vector products (Shi et al., 2025), learning a compact $\mathcal{O}(T)$ configuration aligns directly with the control parameters consumed by Krylov solvers.
2. *Sequence modeling over bandwidth-reducing orderings:* We combine state-space models (Gu & Dao, 2023) with bandwidth-reducing structural permutations (Cuthill & McKee, 1969; Lipton et al., 1979). To our knowledge, no prior preconditioning framework leverages sequence models over structural graph reorderings.
3. *Verification by measurement rather than prediction:* Rather than relying on learned performance surrogates or end-to-end differentiable solver objectives (e.g., Dimola et al., 2025), we verify solver configurations via empirical probe measurement (§3.4) and characterize its operational boundaries (§O.6).

O.3 Sweeping the Probe Budget

The probe budget is a cost knob, not a quality knob. Table 4 evaluates the probe budget across an eightfold range ($T_p \in [10, 80]$). Iteration quality remains virtually invariant: iter-AUC ranges tightly between 535.5 and 536.1, and overall matrix coverage differs by only a single instance, whereas setup time drops by $7\times$ when moving from $T_p = 80$ down to $T_p = 10$. We deploy $T_p = 20$ as our default: it represents the most cost-effective budget that maximizes deep-tolerance convergence (10^{-8} target reached on 96 of 867 matrices—double the deep coverage of $T_p = 80$), requiring only 0.80 s setup time compared to 1.50 s for $T_p = 40$. Evaluation on the 95 large-scale matrices

yields identical conclusions: deep coverage at 10^{-8} remains constant across $T_p \in \{10, 20, 40\}$, with construction costs scaling monotonically. Hence, the probe budget is fixed globally at $T_p = 20$ rather than tuned per matrix scale.

O.4 Isolating the Learned Schedule at Scale

Performance gains are driven by the schedule at large scale as well. To verify causality, we evaluate a zero-learning control—the identical code execution path without the trained model checkpoint—on the same 95 large-scale matrices. The learned configuration achieves 10^{-8} tolerance on 17 matrices versus 14 for the control, with 3 matrices solved exclusively by our learned approach compared to 0 solved exclusively by the control. Loose-band coverage reaches 50 against 45, mean iter-AUC improves by $+29.4$, and time-weighted AUC improves by $+0.41$, with only a single breakdown occurring in each arm. On the pairs where both arms reach the tolerance, the learned configuration arrives first in 89 % of them, and at 10^{-8} it covers 17 matrices against the static bank's 14. Table 17 gives every tolerance.

tolerance	learned	control	learned only	control only	win (%)
10^{-2}	50	45	5	0	80
10^{-3}	36	33	3	0	89
10^{-4}	28	24	4	0	89
10^{-5}	21	18	3	0	95
10^{-6}	18	15	3	0	94
10^{-7}	17	15	2	0	94
10^{-8}	17	14	3	0	94

Table 17: The zero-learning control on the 95 large matrices, one row per tolerance. *learned* and *control* count the matrices each arm drives to the tolerance; *learned only* and *control only* count matrices that only that arm reaches; *win* is the share of decided pairs in which the learned schedule arrives first. The learned schedule covers more matrices at every tolerance and takes most of the decided pairs, and the advantage grows with depth (80 % at 10^{-2} against 94 % at 10^{-8}), while the control reaches no matrix the learned schedule misses.

Iteration efficiency on the small-scale corpus. On the 867 smaller matrices, GMP achieves a slightly higher mean iter-AUC (512.3 vs. 535.5) and broader loose-band coverage (543 vs. 517 at 10^{-2}). However, GMP incurs the precise computational overhead that this paper eliminates: a median per-matrix training time of 133.9 s (431.7 s at p90) compared to our 0.00 s, alongside construction failures on 23 of the 867 matrices (versus 0 for MAMBA-PREC). The trade-offs thus separate cleanly: MAMBA-PREC dominates when solving time is budgeted in seconds, whereas GMP leads when evaluated purely by iteration count. The large-scale benchmark ultimately resolves this trade-off, as per-matrix optimization frequently fails to complete at scale (27 and 43 failures out of 95 instances).

Table 15 (Appendix I) reports the full three-scale encoder ablation.

O.5 Summary of Right-Hand-Side Generalization

The standard evaluation protocol fixes $x = \mathbf{1}$, representing only a single point in the space of right-hand sides. Across six distinct RHS protocols (Appendix O), MAMBA-PREC achieves the top mean iter-AUC and time-weighted AUC in *every* configuration and wins the wall-clock runtime race on 85–100% of pairs against GNP and 90–100% against GMP—all without per-matrix training overhead. Notably, the performance margin is narrowest under the specific protocol for which baselines were originally tuned, confirming that the 20-matrix panel serves as a controlled evaluation setup rather than a direct proxy for the full 867-matrix set.

O.6 Negative Results and Ablation Variants

We evaluated and ultimately rejected three architectural directions: an amortized sparse approximate inverse, unrolled-FGMRES loss objectives incorporating harmonic–Ritz deflation, and a Ja-

cobi anchor. Quantitative comparisons and detailed failure analyses for each variant are provided in Appendix H.

O.7 Paired win rates on the large set, in full

Every paired statistic quoted in §4.2, at the conventions of Table 6 and Table 3:

In paired comparisons over the 95 matrices, the time-to-tolerance win rates across $10^{-2} \dots 10^{-8}$ are $85/82/78/75/86/85/89\,\%$ against GNP, $96/100/100/100/100/100/100\,\%$ against GMP, and $94/92/90/81/78/77/77\,\%$ against AMG. The corresponding iter-AUC win rates are 66.3, 78.9, and $70.5\,\%$, while time-weighted AUC win rates reach 94.7, 97.9, and $95.8\,\%$. All differences are individually statistically significant except for T-AUC (Table 3). The takeaway is straightforward: at this scale, existing learned baselines do not slow down—they fail during construction (GNP fails on $27/95$ matrices and GMP on $43/95$, all occurring at construction time; see Appendix L) or fail to converge. Consequently, our zero-cost amortized configuration is the only learned operator that remains robust at scale. While AMG remains a strong classical reference, reaching the deepest tolerance on 16 of 95 matrices (compared to our 17), it requires $140\times$ our solve time.