ONLY PROJECT ONCE: PROJECTION-ADAPTIVE LOSS FOR EXACT CONSTRAINT SATISFACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Precise constraint satisfaction is a prerequisite to deploying learned models in many areas, motivating methods that repair raw neural predictions with a repair procedure. Current methods unroll multiple repair steps in training and softly penalize constraint violations that remain after the unroll. This is compute- and memory-intensive, lacks robustness when the repair fails to converge, and surrenders most of the constraint satisfaction work to the repair. Our central finding is that, contrary to common practice, a single detached projection step suffices in training. We accomplish this with a Projection-Adaptive Loss (PAL), which uses the constraint residual after this single step to adaptively weigh constraint penalties on the raw prediction. In experiments, PAL is the only method that retains virtually perfect feasibility on extremely nonlinear constraints, and matches or outperforms current methods on synthetic and engineering benchmarks. Because it only requires a single detached projection step, PAL trains 2.5× faster than the canonical repair-based method (DC3) on its own ACOPF benchmark. PAL can also be trained when constraints are expensive to evaluate (e.g., via neural surrogates), a setting where current unrolled methods are memory-intractable.

1 INTRODUCTION

Precise enforcement of hard constraints remains a key challenge in many applications of machine learning. For example, in atmospheric and fluids modeling, neural surrogates must adhere to conservation laws for stable roll-out over long time horizons (Watt-Meyer et al., 2023; Krishnapriyan et al., 2021). In power systems and robotics, learned policies must reliably conform to operational or safety limits to be deployed (Pan et al., 2023; Brunke et al., 2022), and in mechanical and aerospace design, generated designs must not violate stress or aerodynamic limits (Hoyer et al., 2019; Bouhlel et al., 2020). Current methods that enforce constraint satisfaction can typically be divided into two categories. *Penalty methods* use fixed or adaptive constraint weights in the training loss and typically do not exactly satisfy constraints. More recent methods apply a *repair procedure* to raw model predictions to ensure constraints are satisfied.

In this latter category, the current predominant approach to train these methods is to unroll multiple repair steps per training iteration. Any residual violation at the end of the unroll is then softly penalized in the loss with a fixed weight, as shown in Fig. 1.

In our view, this has three major drawbacks: (i) Unrolled repair is compute- and memory-intensive, which limits applicability to relatively cheap-to-evaluate constraints (e.g., closed-form expressions). (ii) Current methods softly penalize post-repair constraint residuals. When the projection iterates do not converge (e.g., under extremely non-linear constraints), a soft penalty does not provide enough training pressure to satisfy these constraints. (iii) We believe that the repair should only act as a final clean-up, and the neural net should learn the bulk of constraint satisfaction. Methods that train with unrolled repair tend to transfer the majority constraint satisfaction work to the repair step.

To address these issues, our contributions are as follows:

1. *Single-Step Repair in Training.* We show that, contrary to the prevailing wisdom, only a *single* detached projection step suffices in training while matching or outperforming multi-step unrolled-repair baselines (§5.1—§5.3). This avoids backpropagation through the repair, and enables tractability in applications where unrolled-repair training is infeasible (§5.2).

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.

2. *Loss Formulation.* We propose a Projection-Adaptive Loss (PAL) which enables reliable training with single-step repair. It is agnostic to the choice of projection step (LM, SQP, Interior-Point, ...). This formulation shifts the feasibility load to the neural net and only treats projection as final clean-up (Fig. 2) and avoids degradation of feasibility when the projector sequence fails to converge (§5.3).

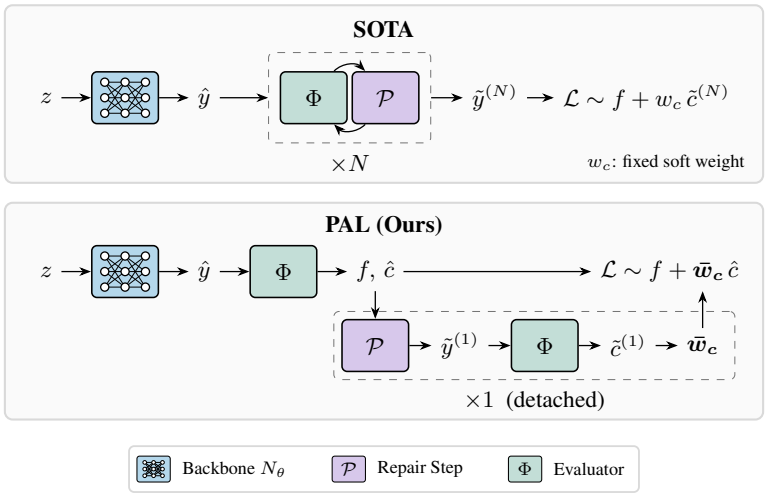


Figure 1: PAL vs. State-of-the-Art (SOTA) repair-aware training. From raw model predictions $\hat{y}$, SOTA iterates projector $\mathcal{P}$ and evaluator Φ for N times per training iteration and then uses a soft penalty on post-repair constraint violation $\tilde{c}^{(N)}$. We instead use a single and detached projection step to $\tilde{y}^{(1)}$. PAL then uses the post-projection residual $\tilde{c}^{(1)}$ to adaptively drive $\bar{w}_c$, which acts as an adaptive penalty on the raw prediction's constraint residual $\hat{c}$.

2 PRELIMINARIES

Problem. We focus on amortized optimization, which targets settings where instances of the same problem family are solved repeatedly under varying conditions where learned models have previously been shown to predict solutions with optimality comparable to classical solvers at many times lower inference cost (Donti et al., 2021; Lastrucci & Schweidtmann, 2025). We consider the constrained optimization problem

$$\min_{y \in \mathbb{R}^n} f(y, z) \quad \text{s.t.} \quad h(y, z) = 0, \qquad g(y, z) \leq 0,$$

where y is the design variable, z an input condition or learned embedding, f the objective, and h and g the equality and inequality constraints. In the following notation, we suppress z for brevity. We write $c(y) \geq 0$ for the vector of constraint violations, with entries $|h_k(y)|$ for the equalities and $\max(g_k(y), 0)$ for the inequalities, and $\mathcal{M} := \{y : c(y) = 0\}$ for the feasible set.

Backbone and Evaluator. A neural network $N_\theta : z \mapsto \hat{y}$ produces predictions, and an evaluator $\Phi(y) \mapsto (f, h, g)$ scores a candidate y by returning the objective and constraint values. We require the evaluator Φ to be differentiable in y.

Projector. A projector $\mathcal{P}$ produces iterates $\tilde{y}^{(i+1)} = \mathcal{P}(\tilde{y}^{(i)}, c(\tilde{y}^{(i)}))$ from $\tilde{y}^{(0)} = \hat{y}$ toward $\mathcal{M}$. We write $\Delta y^{(i)} := \tilde{y}^{(i+1)} - \tilde{y}^{(i)}$ for the repair step's displacement and $R_{i,k} := c_k(\tilde{y}^{(i)})$ for the constraint residuals at iteration i. The projection is computed independently per batch element.

Appendix A provides a full notation overview.

3 Related Work

We focus on settings where Φ is expensive and lacks any exploitable algebraic form, but is smoothly differentiable. Approaches enforcing constraints by construction via hand-specified output layers (Beucler et al., 2021) or differentiable optimization layers (Amos & Kolter, 2017; Agrawal et al., 2019) usually require known structure (linearity, convexity, ...) that Φ does not expose, and are thus out of scope here.

Penalty Methods. A straightforward approach is to incorporate constraints directly into the training loss: $\mathcal{L}_c \sim w_c \cdot c(\hat{y})$, with penalty multipliers w_c either fixed or propagated adaptively (Platt & Barr, 1988; Basir & Senocak, 2023). However, these soft methods often fail to satisfy constraints exactly (Donti et al., 2021). A naive solution is to "bolt on" a repair procedure at test-time. However, this can result in poor solution quality after projection because the trained model fails to anticipate the repair operation (Chu et al., 2026; Grontas et al., 2026), a finding we corroborate in our own experiments. Recent approaches therefore use the repair procedure in end-to-end training. We now discuss prominent variants of this idea.

Repair Methods. Projection-based methods evaluate constraint residuals c and their Jacobian J to perform a linearized quasi-orthogonal projection from $(\tilde{y}^{(i)})$ onto the feasible set $\mathcal{M}$. Because the linearization introduces higher-order error, multiple projection steps are typically used at both training and inference time, often with auto-stopping once feasibility is reached. Recent examples are HardNet (Min & Azizan, 2024) (restricted to linear constraints), ΠNet (Grontas et al., 2026) (restricted to analytically known convex constraints), ENFORCE (Lastrucci & Schweidtmann, 2025), or SnareNet (Chu et al., 2026), which uses Levenberg–Marquardt (LM) projection, improving projection stability when the constraint Jacobian is near-degenerate. We use a similar projector (see Eq. 8), albeit with adaptive damping according to Yamashita & Fukushima (2001). To solve the issue of slow convergence in early training, some methods use adaptive constraint relaxation in early training (Chu et al., 2026), or warm-up using plain penalty methods (Lastrucci & Schweidtmann, 2025). DC3 (Donti et al., 2021) decomposes optimization variables into independent ($\hat{y}_{\text{indep}}$) and dependent (y_{dep}) subsets, predicting the former via the neural backbone and recovering the latter via projection to satisfy equality constraints, with inequalities handled via gradient descent. DC3 requires that equality constraints have known dependence on y_{dep}. Instead of projection, FSNet (Nguyen & Donti, 2025) employs gradient descent to minimize $\|h(y^{(i)})\|^2 + \|g(y^{(i)})\|^2$ at repair step i. FSNet carries formal convergence guarantees, and unlike projection methods, avoids full Jacobian construction, which scales favorably with constraint count. Its core limitation is iteration count. It requires more repair steps N than iterative projection methods, making backpropagation memory-intensive. L-BFGS (Liu & Nocedal, 1989) with early stopping and truncated unrolled differentiation mitigate but do not resolve this.

Gap. Existing methods (DC3, SnareNet, ENFORCE, FSNet, ...) as shown under SOTA in Fig. 1 share three limitations. First, they unroll multiple repair steps per training iteration and backpropagate through them, which is intractable for very expensive evaluators Φ. Second, they train on post-repair constraint residuals using soft penalties with fixed scalar weights w_c, w_d:

$$\mathcal{L} = f(\tilde{y}) + w_d\|\hat{y} - \tilde{y}^{(N)}\|^2 + w_c\|c(\tilde{y}^{(N)})\|^2.$$

When the unrolled projector drives $c(\tilde{y}^{(N)}) \approx 0$ during training, the constraint loss term (w_c) exerts no pressure on the backbone and the repair carries all of the feasibility work as is shown later in Fig. 2. When the unroll does not converge, on the other hand, the residual that remains competes with $f(\tilde{y})$ through the fixed weight w_c, and feasibility degrades as we show in §5.3.

4 Method

We now specify the adaptive penalty update and loss formulation that form our core contribution in detail. The procedure is shown with a damped LM repair step (which is described in more detail in Appendix F, but can be used analogously with SQP and Interior-Point (IP).

Algorithm 1 PAL training.

Require: Backbone N_θ, Evaluator Φ, Epochs T, Hyperparameters $\tau_c, \tau_d, \gamma, D_{\max}, \delta, \eta$

Initialize θ; $\bar{w}_{c,k}, \bar{w}_d \leftarrow 0$.
for $t = 1, \ldots, T$ **do**
  $\hat{y} \leftarrow N_\theta$ ▷ Neural backbone prediction
  $c(\hat{y}),\ f(\hat{y}) \leftarrow \Phi(\hat{y})$ ▷ Evaluator forward pass
  $J \leftarrow \partial c / \partial \hat{y}$ ▷ K reverse-mode VJPs through Φ
  $\tilde{y}^{(1)} \leftarrow \textsc{Project}\big(\hat{y},\ J,\ c(\hat{y});\ \delta\big)$ ▷ Single-step projection with LM, SQP, IP, ...
  $\mathcal{L} \leftarrow \textsc{Loss}\big(\hat{y},\ \tilde{y}^{(1)},\ c(\hat{y}),\ \bar{w}_{c,k},\ \bar{w}_d\big)$ ▷ Eq. 1
  $\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}$ ▷ Training step with LR η
  $c(\tilde{y}^{(1)}) \leftarrow \Phi_c(\tilde{y}^{(1)})$ ▷ Detached evaluator forward
  $\bar{w}_{c,k}, \bar{w}_d \leftarrow \textsc{LogGapUpdate}\big(\bar{w}_{c,k}, \bar{w}_d,\ c(\tilde{y}^{(1)}),\ \tilde{y}^{(1)} - \hat{y};\ \tau_c, \tau_d, \gamma, D_{\max}\big)$ ▷ Eq. 2
end for
return θ

4.1 PAL Training

Algorithm 1 summarizes PAL training. We now discuss the three subroutines PROJECT, LOSS (Eq. 1), and LOGGAPUPDATE (Eq. 2).

Projection. Our core insight is that, in training, it is sufficient to apply a *single* repair step instead of a full unroll. Given a prediction $\hat{y} \sim N_\theta$, we perform a single forward pass on the evaluator, $\Phi(\hat{y}) \mapsto c(\hat{y}), f(\hat{y})$, and K backward passes to assemble the Jacobian J. We then perform a single projection iterate. In our experiments, we use Levenberg–Marquardt with damping according to Yamashita & Fukushima (2001) as the default. PAL is also compatible with other projectors (e.g., Sequential Quadratic Programming (SQP) or Interior-Point (IP) as described in Appendix D). Note that at *test time*, we use up to 10 projection steps with early stopping, just as our baselines do.

Loss. After this single projection step, we compute the training loss as follows

$$\mathcal{L}(\hat{y}, \tilde{y}^{(1)}) = f(\hat{y}) + \sum_{k=1}^{K} \bar{w}_{c,k}\, c_k(\hat{y}) + \bar{w}_d\, \|\Delta y^{(1)}\|, \tag{1}$$

where we only utilize gradient signal from *pre-projection* quantities in training: the pre-projection constraint violations $c(\hat{y})$, displacement $\|\Delta y^{(1)}\| := \|\tilde{y}^{(1)} - \hat{y}\|$ with $\tilde{y}^{(1)}$ detached from the gradient graph, and $f(\hat{y})$. We use $f(\hat{y})$ rather than $f(\tilde{y}^{(1)})$ because it progresses more reliably in early training (see Appendix E for further details).

Projection-Adaptive Penalty Weight Update. To compute the penalty weights $\bar{w}_{c,k}$ and $\bar{w}_d$, we use two post-projection metrics which directly indicate the quality of the repair step:

1. The nonlinear residual after a single projection step $R_{1,k}$. It serves as a proxy for how easy constraint k is to repair. A perfectly linear, uncoupled constraint, for example, can be driven to $R_{1,k} \approx 0$ in a single step. A coupled, nonlinear constraint might, even for small pre-projection violations $c(\hat{y})$, be difficult to project, leaving non-zero $R_{1,k}$.
2. The displacement $\|\Delta y^{(1)}\|$. It is the distance in y the first repair step takes. For the repair to act only as final clean-up, we require $\|\Delta y^{(1)}\|$ to be small.

In training, both these quantities should evolve to be within thresholds τ_d (displacement) and τ_c (constraint residual). To achieve $\|\Delta y^{(1)}\| < \tau_d$ and $R_{1,k} < \tau_c$ for all k, we use the following adaptive penalty scheme:

$$\begin{aligned} \bar{w}_{c,k} &\leftarrow \Big(\bar{w}_{c,k} + \gamma \, \min\big(\log_{10} \tfrac{R_{1,k}}{\tau_c}, \, D_{\max}\big)\Big)_+ \\ \bar{w}_d &\leftarrow \Big(\bar{w}_d + \gamma \, \min\big(\log_{10} \tfrac{\|\Delta y^{(1)}\|}{\tau_d}, \, D_{\max}\big)\Big)_+ \end{aligned} \tag{2}$$

where γ is the penalty growth rate and $D_{\max}$ is the per-step growth cap (in decades). The update acts as follows: if any $R_{1,k} > \tau_c$, i.e., constraint k is hard to project, $\bar{w}_{c,k}$ grows. If $\|\Delta y^{(1)}\| > \tau_d$, i.e., if the repair step displacement is too large, $\bar{w}_d$ grows. On the other hand, where $\|\Delta y^{(1)}\| < \tau_d$ or $R_{1,k} < \tau_c$, the respective weights $\bar{w}_d$ and $\bar{w}_{c,k}$ will gradually decay, freeing learning signal towards the objective. We use log-scaling to make the updates of $\bar{w}_{c,k}$ and $\bar{w}_d$ meaningful across constraints with heterogeneous native scales and to avoid vanishing weight growth for $\tau_{c,k} < R_{1,k} \ll 1$. The growth rate γ sets the adaptation timescale, and $D_{\max}$ promotes stability in early training. We initialize $\bar{w}_d = 0$ and $\bar{w}_{c,k} = 0$ and clip to positive values to prevent rewarding constraint violations. In practice, τ_c is set to the desired constraint tolerance (which is typically known in advance for any application), i.e., we aim for the first projection step to reach feasibility in training. Note that τ_c does not affect the precision reachable at test time. Despite single-step repair in training, constraint violation can be driven to machine precision given enough projection steps, as we show in Appendix G. We characterize the influence of the newly introduced hyperparameters in Ablations (§ 5.4), where a wide band of settings yields comparable performance.

We observe that the per-constraint terms $\bar{w}_{c,k}\, c_k(\hat{y})$ and the displacement term $\bar{w}_d \, \|\Delta y^{(1)}\|$ are often aligned, i.e., reducing one also reduces the other. We show in our Ablations (§ 5.4) that using only one of the two often suffices, but we nevertheless employ both simultaneously as a two-layered safety net, e.g., for stiff constraints ($\|\partial c_k / \partial y\| \gg 1$) for which a small $\|\Delta y^{(1)}\|$ can still leave a large residual $R_{1,k}$ due to damping and nonlinearity. The displacement penalty $\bar{w}_d \|\Delta y^{(1)}\|$ also ensures $f(\hat{y}) \approx f(\tilde{y}^{(1)})$ once the projection step is small, so the backbone's objective is a reliable proxy for the post-projection objective without backpropagating f through the repair step.

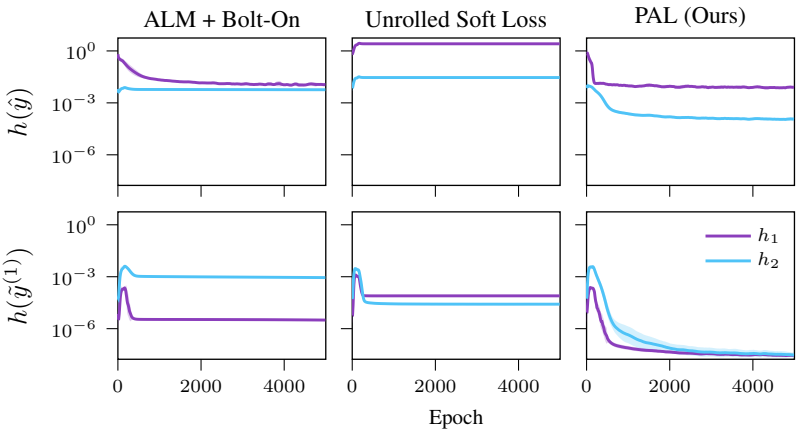


Figure 2: Two-constraint toy problem where h_1 is stiff but linear and therefore trivial to project, and h_2 is non-stiff but nonlinear and difficult to project (more details in Appendix H). **Top:** constraint violations before projection $h(\hat{y})$. **Bottom:** constraint violations after the first projection step $h(\tilde{y}^{(1)})$, equivalent to $R_{1,k}$. **Left:** Adaptive Augmented Lagrangian Method (ALM) (adaptive penalty method) trains without projection and penalizes constraints only according to pre-projection magnitude $h(\hat{y})$, i.e., cannot distinguish constraints by repair difficulty. **Center:** Under soft-loss training (SnareNet, ENFORCE, FSNet, …), the projector drives $h(\tilde{y}^{(N)}) \approx 0$ in training so that constraint pressure on the neural backbone is low, resulting in high violations before and after the first projection step. **Right:** PAL exerts the right amount of training pressure onto $h(\hat{y})$ (and thus the neural backbone) to drive post-projection residuals $h_k(\tilde{y}^{(1)})$ below threshold τ_c (here 10^{-7}).

In Fig. 2, we show how our loss formulation shifts feasibility work between learning (neural backbone N_θ) and repair ($\mathcal{P}$).[1]

5 EXPERIMENTS

Baselines. We compare against ENFORCE (Lastrucci & Schweidtmann, 2025), FSNet (Nguyen & Donti, 2025), SnareNet (Chu et al., 2026), and DC3 (Donti et al., 2021)where applicable. We also include ALM (Basir & Senocak, 2023) (an adaptive penalty method) with Bolt-On repair, i.e., the repair step is only used at test time. The ALM+Bolt-On baseline uses the same LM repair step with adaptive damping as PAL. All other methods use their own repair methods (DC3 uses undamped completion, FSNet uses gradient descent, SnareNet uses damped LM with fixed damping, etc.).

Metrics. All experiments report the *Objective* as $f(\tilde{y}^{(N)})$ and *Feasibility (%)*, the fraction of evaluation samples whose final output $\tilde{y}^{(N)}$ satisfies all constraints to within the benchmark-specific tolerance (typically 10^{-3} or 10^{-4}). To compare compute, we report *Evaluator forward passes* N_Φ^{f} and *Evaluator backward passes* N_Φ^{b}, normalized by batch size.

5.1 COMPARISON ON SYNTHETIC STRESS-TEST BENCHMARKS

In Table 1 we compare results on six benchmarks, each adding a specific stressor. `S1` is the simplest with a single nonlinear equality constraint and a non-convex objective. `S2` adds inequality constraints and a non-convex chained Rosenbrock objective. `S3` features an ill-conditioned objective with constraint coupling. `S4` has nonlinear coupling between two variable blocks. `S5` is an overdetermined nonlinear equality system, whose feasible set reduces to a single point per condition z. `S6` is high-dimensional with $K = 90$ constraints, including redundant inequalities. By construction, projection alone cannot reach the constrained optimum on `S1`–`S4` and `S6`. All synthetic benchmarks have a known constrained optimum $f^\star = 0$. More details on `S1`–`S6` can be found in Appendix I. All methods are hyperparameter-tuned with Bayesian optimization, which is described in further detail in Appendix M.

All baselines reach high feasibility except DC3 due to its undamped completion step, which fails on ill-conditioned problems. With PAL this failure mode can be replicated when the LM projection is used without damping (see App. B).

PAL is competitive with or outperforms repair-aware baselines on optimality across `S1`–`S6`. ENFORCE is the best baseline, except on `S5` where it is structurally inapplicable. ALM+Bolt-On reaches near-perfect feasibility but very poor optimality. Since the projector is held constant between PAL and ALM+Bolt-On, this isolates ALM's lack of repair-awareness as the cause for ALM+Bolt-On's poor performance, which corroborates prior observations that training without repair-awareness can be counterproductive (Chu et al., 2026; Grontas et al., 2026). SnareNet underperforms on objective across the sweep. FSNet performs well, except on `S5`.

Per training step, PAL requires two forward passes through Φ (at $\hat{y}$ and $\tilde{y}^{(1)}$) and $K + 1$ backward passes at $\hat{y}$, of which K build the Jacobian J. In practice, we cache J and reuse it to backpropagate the training loss (Eq. 1) without a second pass through Φ. PAL uses the fewest evaluator passes of the repair-aware methods for problems with few constraints (moderate K) such as `S1`. At, high K (e.g., `S6`) FSNet, which sidesteps Jacobian assembly, requires fewer backward passes. Among projection methods, PAL requires fewest. ALM+Bolt-On is cheapest overall but achieves poor results. Appendix O elaborates further on compute scaling.

5.2 CASE STUDIES IN SCIENCE AND ENGINEERING

Table 2 shows results for three engineering case studies. `E1` optimizes aircraft designs using expensive neural Signed Distance Function (SDF) geometry representation with surrogates for structural properties and aerodynamics (Sung et al., 2025). `E2` (urban planning) uses a video-diffusion model (Perini

[1] Side note on representation: We hypothesize that pushing the neural backbone N_θ to predict near-feasible $\hat{y}$ could also encourage the learned embeddings of N_θ to approximately encode the constraint manifold $\mathcal{M}$, a property other repair-aware losses do not actively encourage.

Table 1: Synthetic stress-test experiments. Means and standard deviations across 10 seeds. "—": failure on 2 or more seeds. Forward and backward evaluator passes N_ϕ^f and N_ϕ^b per training step are only shown for extreme cases `S1` ($K = 1$) and `S6` ($K = 90$), with full results in Table 12.

	Bench.	ALM+ Bolt-On	ENFORCE	DC3	FSNet	SnareNet	PAL [Ours]
Feas. (%)	S1	100 ± 0	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	S2	100 ± 0	100 ± 0	97 ± 7	100 ± 0	100 ± 0	100 ± 0
	S3	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0
	S4	100 ± 1	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	S5	100 ± 0	—	—	51 ± 31	100 ± 0	100 ± 0
	S6	100 ± 0	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	Mean	**100**	—	—	92	**100**	**100**
Obj.	S1	0.07 ± 0.12	0.00 ± 0.00	—	0.00 ± 0.00	0.78 ± 0.10	0.03 ± 0.04
	S2	0.27 ± 0.17	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.16 ± 0.00	0.00 ± 0.00
	S3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.00 ± 0.00
	S4	19.47 ± 5.25	0.01 ± 0.01	—	0.00 ± 0.00	0.09 ± 0.01	0.00 ± 0.00
	S5	0.00 ± 0.00	—	—	4.41 ± 2.16	0.00 ± 0.00	0.00 ± 0.00
	S6	0.14 ± 0.03	0.00 ± 0.00	—	0.00 ± 0.00	0.60 ± 0.06	0.03 ± 0.01
	Mean	3.32	—	—	0.73	0.27	**0.01**
N_Φ^f	S1	1.0 ± 0.0	6.2 ± 0.1	—	72 ± 77	35.0 ± 5.0	2.0 ± 0.0
	S6	1.0 ± 0.0	7.1 ± 0.1	—	15.5 ± 0.0	81.1 ± 2.9	2.0 ± 0.0
N_Φ^b	S1	1.0 ± 0.0	4.1 ± 0.1	—	11.0 ± 5.6	33.6 ± 5.0	2.0 ± 0.0
	S6	1.0 ± 0.0	128 ± 2	—	7.2 ± 0.0	253 ± 12	91.0 ± 0.0

et al., 2026) to evaluate pedestrian wind comfort in cities. Both of these benchmarks are compute-intensive. `E3` is a variant of the AC Optimal Power Flow (ACOPF) benchmark that motivated DC3 Donti et al. (2021), but our variant has more constraints (576 vs. 256). This benchmark is difficult due to the high number of constraints and nonlinear, rank-deficient equalities. Detailed descriptions of `E1`—`E3` can be found in Appendix K.

On both `E1` and `E2`, the unroll-repair baselines are Out Of Memory (OOM) even at a mini-batch size of 1, except SnareNet on `E1` which OOM's at a mini-batch size of 8. With our 4×GH200, PAL can be used with batches of 32 on `E1`, and 16 on `E2`. On `E1` and `E2`, the penalty methods are the least memory-hungry, but fall short in optimality and feasibility, respectively. On `E3`, DC3's flagship benchmark, PAL leads in both feasibility and optimality while training $2.5\times$ faster. We reproduce the violation magnitudes of DC3 within their reported precision (Donti et al., 2021). On `E3`, the penalty-based methods train in minutes because they avoid K backward passes, but fall short catastrophically in constraint satisfaction.

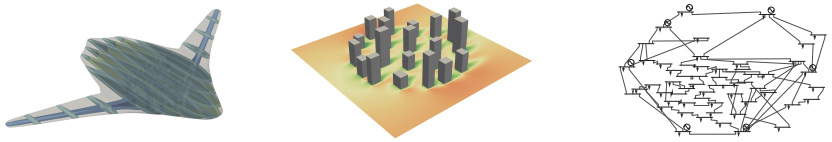
Overall, PAL is the only method that combines high feasibility with a competitive objective on all three engineering benchmarks. Regarding the lacking robustness of unroll baselines, we hypothesize on the varying causes of failures (IPOPT on `E2` and ENFORCE, FSNet and SnareNet on `E3`) in Appendix L. These results also show that, as one might expect, unrolled-repair methods are not tractable on problems with expensive constraint simulators (`E1` and `E2`).

5.3 Highly Nonlinear Constraints

Figure 3 shows how PAL and unrolled soft-loss baselines behave under extreme constraint nonlinearity on a synthetic problem with tunable curvature κ. In addition to feasibility and optimality, we report the ratio $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$ in late training. It represents the share of overall training pressure exerted by the constraint loss. Details on the problem setup and full results are shown in Appendix C.

PAL reaches optimality on par with the full-unroll methods (Fig. 3b) and retains the highest feasibility of all methods at extreme curvature of $\kappa = 10^6$ (Fig. 3a). PAL achieves this because its adaptive weight update applies increasing pressure to reduce pre-projection constraints when the linear first projection step leaves large residuals, as shown in Fig. 3c. The soft-loss baselines do not possess this mechanism: at high curvature, when the unrolled projection steps alone fail to converge to feasible solutions, constraints and optimality compete on equal footing due to the fixed-weight summed loss.

Table 2: Case studies in science and engineering. Mem: peak GPU memory (GB) at `batchsize=1`. Time: training wall-clock. 10 seeds each, IPOPT 5 restarts × 64 queries. $*$: Maximum constraint violation. Bold marks the best value among feasible methods. ‡: 49 GB on 3 seeds and >96 GB (OOM) on 7 seeds in the first two training epochs. §: `batchsize=8`. †: 0% feasibility. △: Non-convergence. ◇: Structurally inapplicable. Baseline failure modes are discussed in Appendix L.



	E1 Unmanned Aircraft			**E2 Urban Planning**			**E3 ACOPF (IEEE-57)**		
Method	$\text{Obj}_{\times 10^{-2}}$	Feas%	Mem	$\text{Obj}_{\times 10^{1}}$	Feas%	Mem	$\text{Obj}_{\times 10^{4}}$	$\text{Viol}^{*}_{\times 10^{-2}}$	Time
PAL	$\mathbf{-3.8\pm0.1}$	$\mathbf{100\pm0}$	4.4	-2.71 ± 0.02	$\mathbf{98\pm3}$	25.7	$\mathbf{4.28\pm0.05}$	$\mathbf{0.15\pm0.08}$	**3h50m**
ALM	-2.6 ± 1.5	99 ± 1	**4.1**	$\mathbf{-2.74\pm0.01}$	58 ± 33	**15.7**	$3.60\pm2.00^{\dagger}$	1100 ± 731	$1\text{m}18\text{s}^{\dagger}$
ALM+Bolt-On	-2.6 ± 1.5	$\mathbf{100\pm0}$	**4.1**	-2.73 ± 0.01	69 ± 25	**15.7**	$4.52\pm0.26^{\dagger}$	38 ± 56	$1\text{m}18\text{s}^{\dagger}$
DC3	◇	◇	◇	◇	◇	◇	4.34 ± 0.06	0.53 ± 0.23	9h52m
FSNet	OOM	OOM	> 96	OOM	OOM	> 96	$1.15\pm0.46^{\dagger}$	$> 10^{17}$	$3\text{h}29\text{m}^{\dagger}$
SnareNet	OOM	OOM	$> 96^{\S}$	OOM	OOM	$> 49^{\ddagger}$	△	△	△
ENFORCE	OOM	OOM	> 96	OOM	OOM	> 96	△	△	8h54m
IPOPT	-3.7 ± 0.1	**100**	—	△	△	△	see Donti et al. (2021)		

This can be seen also by the lower $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$ in Fig. 3c. For PAL at $\tau_c = 1$, the adaptive weight growth is effectively disabled and we observe similar degradation.

These results demonstrate that, despite the single-step training, PAL is **not** limited to near-linear constraints. In fact, it is more robust in cases where unrolled projection alone is not sufficient for reliable constraint satisfaction due to the use of our projection-adaptive weight loss.

5.4 Ablation Studies

In Appendix B, we ablate PAL's loss and weight update mechanisms and their hyperparameters. Slowing the adaptive weight growth via γ or $D_{\max}$ (recall Eq. 2 for definitions) can improve optimality, while an excessive growth rate γ can collapse optimality unless mitigated by low $D_{\max}$. We find that outcomes are insensitive to the threshold τ_d across several orders of magnitude. If τ_d is set to a sufficiently small numerical value, further tightening has no effect. Using $f(\tilde{y}^{(1)})$ in the training loss in place of $f(\hat{y})$ degrades both optimality and feasibility. We hypothesize on a potential cause in detail in Appendix E. Disabling either $\bar{w}_c$ or $\bar{w}_d$ preserves performance, suggesting redundancy of these loss terms. Disabling both collapses optimality and degrades pre-projection feasibility, yet the projector alone still recovers $\sim$99% feasibility. This validates our load-share

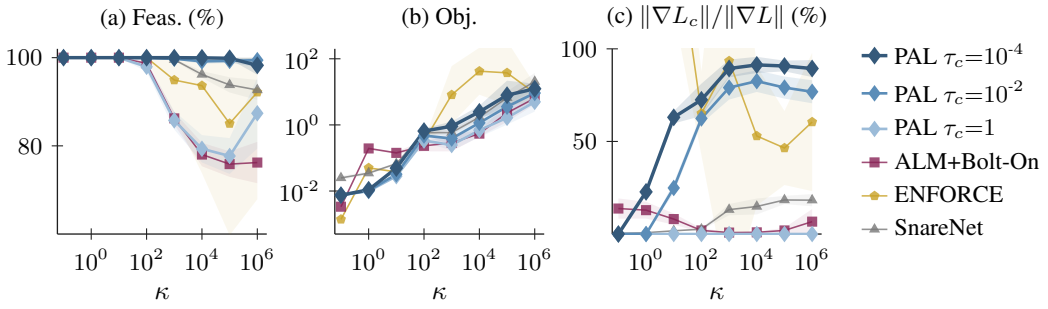


Figure 3: Feasibility, objective and constraint training pressure $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$ over curvature κ. PAL retains best-in-class feasibility at high curvature despite a single repair step, because its adaptive loss raises training pressure in response to first-step residuals.

argument: the projector achieves feasibility in isolation, but ideal results require non-trivial constraint pressure on the backbone.

Alternative Repair Steps. In Appendix D we compare PAL with SQP and Interior-Point repair in place of LM. Both roughly match LM across the synthetic benchmarks, showing that PAL is agnostic to the type of projection step, as long as it is robust on ill-conditioned Jacobians. (Newton projection, for example, which is a special case of LM without damping, is not. We discuss damping hyperparameters of the LM step in Appendix B).

Repair Steps in Inference. In Appendix G we vary the number of repair steps N at test time and show that single-step training does not limit the precision of constraint satisfaction PAL reaches at inference.

6 Conclusion

Contributions and Outcome. We introduced PAL, a scheme where single-step projection metrics drive adaptive constraint weights. We show that with this method, fully unrolled repair is not necessary to achieve good feasibility or optimality, and a single step is in fact sufficient. In fact, PAL is the *only* method that achieves best-in-class results simultaneously on synthetic stress-testing (Tab. 1), engineering applications (Tab. 2), and under extremely high curvature (Fig. 3).

In §5.2 we show that using only a single detached repair step enables tractable training when constraints are expensive to evaluate, like with high-fidelity physics surrogates. Current full-unroll methods were not able to train on these problems due to memory. Using only a single detached step also results in fewer constraint evaluator calls (§5.1), and PAL trains 2.5× faster than DC3 on its flagship benchmark (while still providing better feasibility and optimality).

Lastly, PAL's adaptive weight scheme shifts feasibility towards the neural backbone (Fig. 2) which soft-loss unrolled baselines do not do. Under extremely non-linear constraints, this manifests in PAL retaining near-full feasibility while unrolled soft-loss baselines degrade (§5.3).

Limitations. Several limitations remain: (1) Like all other projection-based methods, we require the constraint evaluation to be differentiable. (2) In all repair-based methods, feasibility is defined w.r.t. the evaluator Φ. If Φ is inaccurate, constraints will be satisfied according to an incorrect reference. (3) Assembly of the Jacobian requires K backward passes through the evaluator Φ for all projection-based methods. PAL alleviates this issue somewhat as it only requires a single projection step, but problems where the number of constraints K is extremely large are still compute-intensive. And (4), PAL introduces several hyperparameters (γ, τ_d, τ_c, $D_{\max}$, δ, $\lambda_{\min}$), though in practice τ_c follows immediately from problem-specific tolerances, $D_{\max}$ can be left at a conservative default, and δ and $\lambda_{\min}$ have negligible impact.

Future Work. A first natural extension is to study how PAL performs on tasks beyond amortized optimization, where the objective f is a data-driven loss (e.g., MSE). Second, efficiency improvements or auto-calibration of the remaining hyperparameters could make single-step repair with PAL even more easily usable in scientific and engineering applications. Examples include estimating $\bar{w}_d$ and $\bar{w}_c$ from sparse projection only every few steps, or auto-calibrating γ and $D_{\max}$ from early-trajectory metrics or τ_d from the relative error between $f(\hat{y})$ and $f(\tilde{y}^{(1)})$). Third, since the repair step is never backpropagated through, PAL does not require a differentiable projector. This opens the door to bespoke, non-differentiable repair methods that are specialized to niche applications beyond the projection methods explored here.

AI use statement

We used LLMs in code generation to set up and run experiments, for brainstorming and discussing ideas, for searching existing literature, and to interpret diagnostics and results. The paper content is human-written. LLMs were used for minor text improvements and for creating and formatting tables and figures, always under close supervision by us. The authors take full responsibility for all AI-generated work.

Ethics statement

We see no ethical concerns beyond those of machine learning research in general.

Reproducibility statement

Our paper states the method in detail, and Appendices F, I, J, K, L, M, and N provide additional details to reproduce our results. Source code, including SLURM scripts to run the experiments with exactly the same configurations as we did, will be released publicly with the camera-ready version and is provided to the reviewers as supplementary material.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

AI4OPT. ML4OPF: Machine learning for optimal power flow. `https://github.com/AI4OPT/ML4OPF`, 2024.

Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 136–145, 2017.

Shamsulhaq Basir and Inanc Senocak. An adaptive augmented Lagrangian method for training physics and equality constrained artificial neural networks. *arXiv preprint arXiv:2306.04904*, 2023.

Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters*, 126(9):098302, 2021. doi: 10.1103/PhysRevLett.126.098302.

Mohamed Amine Bouhlel, Sicheng He, and Joaquim R. R. A. Martins. Scalable gradient-enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes. *Structural and Multidisciplinary Optimization*, 61(4):1363–1376, 2020. doi: 10.1007/s00158-020-02488-5.

Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444, 2022. doi: 10.1146/annurev-control-042920-020211.

Ya-Chi Chu, Alkiviades Boukas, and Madeleine Udell. SnareNet: Flexible repair layers for neural networks with hard constraints. *arXiv preprint arXiv:2602.09317*, 2026.

Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations (ICLR)*, 2021.

Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. doi: 10.1137/S0036144504446096.

Panagiotis D. Grontas, Antonio Terpin, Efe C. Balta, Raffaello D'Andrea, and John Lygeros. ΠNet: Optimizing hard-constrained neural networks with orthogonal projection layers. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, Guy Shiran, Nir Zabari, Ori Gordon, Poriya Panet, Sapir Weissbuch, Victor Kulikov, Yaki Bitterman, Zeev Melumian, and Ofir Bibi. LTX-Video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2024.

Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019. NeurIPS 2019 Workshop on Solving Inverse Problems with Deep Networks.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26548–26560, 2021.

Giacomo Lastrucci and Artur M. Schweidtmann. ENFORCE: Nonlinear constrained learning with adaptive-depth neural projection. *arXiv preprint arXiv:2502.06774*, 2025.

Richard Lincoln. PYPOWER: Port of MATPOWER to Python. `https://github.com/rwl/PYPOWER`, 2011.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1–3):503–528, 1989. doi: 10.1007/BF01589116.

Uday Mallappa, Hesham Mostafa, Mikhail Galkin, Mariano Phielipp, and Somdeb Majumdar. FloorSet – a VLSI floorplanning dataset with design constraints of real-world SoCs. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2024. doi: 10.1145/3676536.3676814.

Robert A. McDonald and J. R. Gloudemans. Open vehicle sketch pad: An open source parametric geometry and analysis tool for conceptual aircraft design. In *AIAA SciTech 2022 Forum*, 2022. doi: 10.2514/6.2022-0004.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

Youngjae Min and Navid Azizan. HardNet: Hard-constrained neural networks with universal approximation guarantees. *arXiv preprint arXiv:2410.10807*, 2024.

Hoang T. Nguyen and Priya L. Donti. FSNet: Feasibility-seeking neural network for constrained optimization with guarantees. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, 2 edition, 2006. ISBN 978-0-387-30303-1.

Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H. Low. DeepOPF: A feasibility-optimized deep neural network approach for AC optimal power flow problems. *IEEE Systems Journal*, 17(1): 673–683, 2023. doi: 10.1109/JSYST.2022.3201041.

Janne Perini, Rafael Bischof, Moab Arar, Ayça Duran, Michael A. Kraus, Siddhartha Mishra, and Bernd Bickel. Pretrained video models as differentiable physics simulators for urban wind flows. *arXiv preprint arXiv:2603.21210*, 2026.

John C. Platt and Alan H. Barr. Constrained differential optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1988.

C. Bane Sullivan and Alexander A. Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37): 1450, 2019. doi: 10.21105/joss.01450.

Nicholas Sung, Steven Spreizer, Mohamed Elrefaie, Matthew C. Jones, and Faez Ahmed. BlendedNet++: A large-scale blended wing body aerodynamics dataset and benchmark. *arXiv preprint arXiv:2512.03280*, 2025.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun. pandapower — an open-source python tool for convenient modeling, analysis, and optimization of electric power systems. *IEEE Transactions on Power Systems*, 33(6):6510–6521, 2018. doi: 10.1109/TPWRS.2018.2829021.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. doi: 10.1007/s10107-004-0559-y.

Oliver Watt-Meyer, Gideon Dresdner, Jeremy McGibbon, Spencer K. Clark, Brian Henn, Matthew E. Peters, Christopher S. Bretherton, James Duncan, Noah D. Brenowitz, Karthik Kashinath, Michael S. Pritchard, and Boris Bonev. ACE: A fast, skillful learned global atmospheric model for climate prediction. *arXiv preprint arXiv:2310.02074*, 2023.

Nobuo Yamashita and Masao Fukushima. On the rate of convergence of the Levenberg-Marquardt method. In G. Alefeld and X. Chen (eds.), *Topics in Numerical Analysis*, volume 15 of *Computing Supplementa*, pp. 239–249. Springer, Vienna, 2001. doi: 10.1007/978-3-7091-6217-0_18.

Dominic Yang, Prasanna Balaprakash, and Sven Leyffer. Modeling design and control problems involving neural network surrogates. *Computational Optimization and Applications*, 83:759–800, 2022. doi: 10.1007/s10589-022-00404-9.

Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011. doi: 10.1109/TPWRS.2010.2051168.

A Notation Summary

See Table 3.

Table 3: Notation summary.

Symbol	Meaning
$y,\ \hat{y},\ \tilde{y}^{(i)}$	design variable, raw neural backbone prediction, i-th projection iterate
z	input condition or latent embedding
$f,\ h,\ g$	objective, equality constraints, inequality constraints
$c_k,\ c_{\mathcal{A}}$	per-constraint residual, active residual vector
$\mathcal{A}$	active index set
$J_{\mathcal{A}}$	active constraint Jacobian $\partial c_{\mathcal{A}}/\partial y$
$\mathcal{M}_{\mathcal{A}}$	local feasible manifold $\{y : c_{\mathcal{A}}(y) = 0\}$
N	number of projection steps
$\Delta y^{(i)},\ R_i$	projection step displacement, constraint violation residual at iterate i
$\lambda_i,\ \delta$	LM damping at step i, damping hyperparameter
$\tau_c,\ \tau_d$	residual threshold, displacement threshold
$\bar{w}_{c,k},\ \bar{w}_d$	adaptive constraint and displacement penalty weights
$\gamma,\ D_{\max}$	log-gap rate, per-step growth ceiling (decades)

B Ablation Study

In this appendix, we report full ablations for PAL. These ablations were carried out on the six synthetic benchmarks `S1`–`S6`, over $n = 10$ seeds each. We report worst-bench (i.e., maximum) objective across the six synthetic benchmarks and minimum feasibility across the six benchmarks. Additionally, we report the pre-projection residual $\|c(\hat{y})\|_\infty$ averaged across `S1`–`S6`. This is a measure of how much of the constraint-satisfaction load the backbone N_θ carries. Table 4 shows the full ablation summary.

Table 4: Ablation of PAL hyperparameters on synthetic benchmarks ($n = 10$ seeds). Blank cells inherit the `Base` value. Bold = best per column. The f column lists the argument used inside the objective term of the training loss ($\hat{y}$ vs $\tilde{y}^{(1)}$). `Dec.` marks whether the multiplier rule's decay half is active: ✓ = bidirectional (default), 7 = monotone-only growth.

ID	δ	$\lambda_{\min}^{\text{proj}}$	γ	$D_{\max}$	τ_c	τ_d	LR	f	Dec.	Obj. (max) ↓	Feas. (%) ↑	$\|c(\hat{y})\|_\infty$ ↓
Base	10^{-3}	10^{-6}	10^{-2}	1	10^{-4}	10^{-4}	10^{-4}	$\hat{y}$	✓	0.08 ± 0.01	$100 \pm 0\%$	0.06 ± 0.02
A	0									0.07 ± 0.01	$100 \pm 0\%$	0.06 ± 0.02
B	10^{-4}									0.07 ± 0.01	$100 \pm 0\%$	$\mathbf{0.05 \pm 0.01}$
C	10^{-2}									0.10 ± 0.02	$100 \pm 0\%$	0.06 ± 0.02
D	0	0								0.11 ± 0.09	$100 \pm 0\%$	0.06 ± 0.02
E			10^{-3}							0.06 ± 0.03	$100 \pm 0\%$	0.09 ± 0.06
F			10^{-1}							0.72 ± 0.04	$100 \pm 0\%$	0.06 ± 0.02
G			10^{-1}	0.1						0.08 ± 0.01	$100 \pm 0\%$	0.06 ± 0.02
H				0.2						0.04 ± 0.03	$100 \pm 0\%$	0.06 ± 0.04
J				5						0.23 ± 0.02	$100 \pm 0\%$	0.06 ± 0.02
K						10^{-2}				0.08 ± 0.01	$100 \pm 0\%$	0.07 ± 0.02
L							10^{-3}			0.13 ± 0.02	$100 \pm 0\%$	0.22 ± 0.06
M						10^{-4}		$\tilde{y}^{(1)}$		0.84 ± 0.79	$99 \pm 1\%$	0.40 ± 0.01
N						10^{-1}		$\tilde{y}^{(1)}$		0.84 ± 0.79	$99 \pm 1\%$	0.61 ± 0.06
O						10^{-1}		$\hat{y}$		$\mathbf{0.03 \pm 0.04}$	$100 \pm 0\%$	0.14 ± 0.04
P					10^{10}					0.08 ± 0.03	$100 \pm 0\%$	0.07 ± 0.02
Q						10^{10}				0.06 ± 0.01	$100 \pm 0\%$	0.27 ± 0.02
R					10^{10}	10^{10}				0.93 ± 0.52	$99 \pm 2\%$	1.19 ± 0.11
S									✗	0.09 ± 0.01	$100 \pm 0\%$	$\mathbf{0.05 \pm 0.01}$

Projector Damping (A–D). Rows A–D show that performance is insensitive to the residual-scaling coefficient δ on these tasks. Rows A, B, C sweep δ over three orders of magnitude and match `Base` on objective and feasibility. Row A ($\delta = 0$) retains the numerical floor $\lambda_{\min}^{\text{proj}}$. Row D removes both residual-scaled damping and the floor, but on S2–S6 the masked $BB^\top$ is singular, the Cholesky solve fails, and the eigenvalue fallback of the projector applies its own floor of 10^{-4}. Only S1 runs undamped, and there two of ten seeds degrade in objective, which accounts for the larger spread of row D. A fully undamped step is exposed on ill-conditioned $BB^\top$, which is the failure mode of the undamped completion of DC3 in Table 1.

Multiplier-Update Timescale (E–J). The growth rate γ and the per-step cap $D_{\max}$ jointly control how aggressively the constraint weights $\bar{w}_{c,k}$ react to residual overshoot. Lowering γ marginally improves optimality (E), since slower weight growth leaves more early capacity for the objective. Raising γ collapses optimality (F) as the penalty term overwhelms f. Pairing a high γ with a low $D_{\max}$ (G) recovers `Base` performance, confirming that γ and $D_{\max}$ act together rather than independently. Tightening $D_{\max}$ alone (H) halves `Base`'s worst-bench objective, while loosening it (J) is several times worse but not catastrophic.

Tolerances and Learning Rate (K–L). The displacement threshold τ_d can be loosened by two orders of magnitude without effect (K). It only needs to be "small enough". The learning-rate sensitivity in L is generic to neural training and not specific to PAL.

Pre-Projection Objective (M–O). Replacing $f(\hat{y})$ with $f(\tilde{y}^{(1)})$ in the training loss degrades both optimality and feasibility across two values of τ_d (M, N), while a matched control with the same τ_d and $f(\hat{y})$ (O) recovers near-baseline. We hypothesize this is related to how gradients flow back through the projection step, which we detail in Appendix E.

Load-Share (P–R). Disabling either $\bar{w}_c$ or $\bar{w}_d$ alone (by setting its threshold to 10^{10}) preserves objective and feasibility on this benchmark (P, Q), with the only visible effect being a higher pre-projection $\|c(\hat{y})\|_\infty$ in Q where $\bar{w}_d$ is off. The two terms in Eq. 2 therefore act as redundant safety nets here, although both may be needed for stiffer constraint systems. Disabling both (R) lifts all constraint pressure from the backbone: optimality collapses and pre-projection $\|c(\hat{y})\|_\infty$ blows up,

yet the projector still recovers $\sim 99\%$ feasibility on its own. This is an empirical demonstration of our load-share argument: the projector is often capable enough to handle feasibility in isolation, but a more preferable outcome is for the neural backbone to carry significant constraint-satisfaction burden, and the projector (or, more generally: the repair step) to be utilized as clean-up only.

Together, the ablation isolates the design choices behind PAL's loss formulation: the log-gap weight update with a growth cap for the multipliers (E–J), the pre-projection objective $f(\hat{y})$ (M–O), and load-share, where backbone pressure remains necessary for optimality even when the projector alone recovers feasibility (P–R).

C Constraint Curvature Ablation

Here, we compare how PAL and the baselines perform under extreme constraint curvature. As a benchmark, we construct a problem with a single equality constraint, whose curvature κ ranges from 0 to 10^6. The problem dimensionality is $\dim(y) = 8$. Equation 3 shows the constraint $c(y)$ (variable-frequency sine on the hyperplane $a^\top y = 0$) with a fixed orthonormal pair $a \perp b$, drawn once and shared across all κ and all seeds.

$$c(y) \;=\; a^\top y \;-\; \sin\!\big(\omega\, b^\top y\big), \qquad \kappa \;=\; \omega^2, \qquad f_x(y) \;=\; \big\|\, T^{-1}(y) - T^{-1}(x) \,\big\|_2^2. \tag{3}$$

The parameter κ is the curvature stemming from the high-frequency sine component of the constraint. The objective is formulated such that it has a unique optimum (non-multimodal) and a closed-form $f^\star$ at every κ, using the shear

$$T(v) = v + a\sin\!\big(\omega\, b^\top v\big), \qquad T^{-1}(v) = v - a\sin\!\big(\omega\, b^\top v\big). \tag{4}$$

All methods use bayesian-optimization tuned hyperparameters (see Appendix M). Plain ALM is 0% feasible at all κ, so we omit it. We use 10 seeds and 512 evaluations per seed, and all runs are in double precision. We measure feasibility here as violations below 10^{-4}.

Additionally, we report the ratio $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$, which is the recorded mean over the last five logged epochs of each training seed. This ratio is effectively the overall share of training pressure exerted by the constraint violation.

Table 5 shows feasibility, optimality gap, and constraint training pressure for all methods at varying κ. At zero curvature (purely linear constraint), all methods reach 100% feasibility with virtually no optimality gap. For PAL at $\kappa = 0$, the constraint pressure $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$ is equal to zero across $\tau_c \in \{10^{-4}, 10^{-2}, 1\}$. This is because the purely linear constraint at $\kappa = 0$ is repaired exactly with a single projection step, so there is no single-step remainder.

As can be expected, the constraint training pressure in PAL increases with higher curvature κ (due to larger post-step residuals R_1) and with lower residual targets τ_c, reaching up to $\|\nabla_\theta L_c\|/\|\nabla_\theta L\| = 91\%$ at $\kappa = 10^4$ and $\tau_c = 10^{-4}$. This is in contrast to the full-unroll baselines. Here, training pressure is weaker at high curvature because post-repair residuals are soft-penalized (or, in the case of FSNet, not at all) instead of being driven by adaptive weight growth as in PAL. One exception is ENFORCE (which, through the soft displacement penalty with constant w_d, exerts pressure especially at low κ where displacement $\|\tilde{y} - \hat{y}\|$ is large). But also for ENFORCE, constraint pressure, feasibility and optimality degrade at higher curvatures.

Overall, this experiment demonstrates that single-step repair in training does **not** limit PAL to near-linear constraints. This is by design, as the single-step residual under high curvature triggers improved load sharing in PAL. On the contrary, thanks to the use of adaptive weight updates, PAL exhibits **better** performance than existing full-unroll baselines under high-curvature constraints.

All runs use double precision, since above $\kappa = 10^4$ the constraint Jacobian is large even at small constraint values, and in single precision the rounding error in y alone produces violations above the feasibility threshold. DC3 does not run with its undamped Newton completion, which diverges on ill-conditioned Jacobians. We instead use a damped Newton completion with Yamashita-Fukushima damping.

Table 5: Full results for high-curvature experiment. κ from 0 to 10^6, 10 seeds, all methods trained and evaluated in `float64`. $\|\nabla_\theta L_c\|/\|\nabla_\theta L\|$ is the share of constraint training pressure. Bold marks the best feasibility at each κ across all methods. †: gradient ratio carried over from the source ALM training run. Plain ALM without Bolt-On is omitted because it is 0% feasible at every κ. DC3 uses a Newton completion with Yamashita-Fukushima damping, since the undamped completion diverges for every $\kappa > 0$.

	PAL (τ_c=10^{-4})			PAL (τ_c=10^{-2})		
κ	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)
κ=0	**100±0**	0.01±0.00	0.0	**100±0**	0.01±0.00	0.0
κ=1	**100±0**	0.01±0.00	23	**100±0**	0.01±0.00	0.0
κ=10	**100±0**	0.05±0.01	63	**100±0**	0.03±0.00	25
κ=10^2	**100±0**	0.66±0.13	72	99.9±0.1	0.48±0.05	62
κ=10^3	**100±0**	0.96±0.40	89	99.8±0.2	0.42±0.20	79
κ=10^4	**100.0±0.1**	2.69±1.10	91	99.2±0.5	1.23±0.52	83
κ=10^5	**99.9±0.2**	11.84±9.98	91	99.3±0.5	4.22±2.60	79
κ=10^6	98.3±2.1	13.16±4.98	89	**99.4±1.5**	10.50±6.20	77
	PAL (τ_c=1)			ALM+Bolt-On		
κ	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)
κ=0	**100±0**	0.01±0.00	0.0	**100±0**	0.00±0.00	14†
κ=1	**100±0**	0.01±0.00	0.0	**100±0**	0.19±0.01	13†
κ=10	**100±0**	0.03±0.00	0.0	**100±0**	0.14±0.01	8.1†
κ=10^2	97.9±1.1	0.34±0.03	0.0	98.7±0.7	0.25±0.08	1.9†
κ=10^3	85.8±2.1	0.26±0.10	0.0	86.2±1.4	0.34±0.22	0.8†
κ=10^4	79.3±3.1	0.78±0.43	0.0	78.0±2.7	0.56±0.12	0.9†
κ=10^5	77.7±4.1	1.80±0.75	0.0	75.9±2.9	2.71±1.50	1.9†
κ=10^6	87.4±8.0	6.12±4.17	0.0	76.2±4.7	6.97±2.78	6.7†
	DC3			ENFORCE		
κ	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)
κ=0	**100±0**	0.76±0.03	114	**100±0**	0.00±0.00	234
κ=1	76.4±1.8	0.86±0.03	143	**100±0**	0.05±0.00	341
κ=10	75.6±2.5	0.99±0.05	125	**100±0**	0.04±0.00	170
κ=10^2	89.7±2.0	1.26±0.06	88	**100±0**	0.32±0.02	65
κ=10^3	90.4±2.0	1.20±0.22	92	94.9±8.3	37.90±58.84	93
κ=10^4	87.5±3.4	1.47±0.70	94	93.7±19.9	233±610	53
κ=10^5	91.5±5.7	3.12±0.46	94	85.1±25.3	393±982	46
κ=10^6	92.2±8.4	5.33±0.74	93	92.2±24.3	10.95±7.14	61
	FSNet			SnareNet		
κ	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)	Feas. (%)	Obj.	$\frac{\|\nabla L_c\|}{\|\nabla L\|}$ (%)
κ=0	**100±0**	0.00±0.00	0.0	**100±0**	0.03±0.00	0.4
κ=1	**100±0**	0.00±0.00	0.0	**100±0**	0.03±0.00	0.7
κ=10	96.7±9.3	72.83±52.42	0.0	**100±0**	0.07±0.00	1.7
κ=10^2	96.0±11.5	86.24±29.21	0.0	**100±0**	0.58±0.08	2.6
κ=10^3	69.6±47.8	88.06±20.81	0.0	99.6±0.2	0.60±0.10	13
κ=10^4	51.5±27.1	82.84±26.01	0.0	96.2±0.8	1.73±0.30	15
κ=10^5	39.4±38.1	104±28	0.0	93.8±1.7	7.22±2.99	18
κ=10^6	17.1±34.3	107±26	0.0	92.7±2.3	23.67±12.51	18

D Alternative Repair Steps

We swap the damped LM projector of Equation 8 for two standard single-step repair operators, an Sequential Quadratic Programming (SQP) step with elastic slacks (Nocedal & Wright, 2006; Gill et al., 2005) and an Interior-Point (IP) step (Nocedal & Wright, 2006; Wächter & Biegler, 2006).

Table 6: Objective at convergence ($f^\star = 0$) for the three repair step mechanisms on the synthetic benchmarks `S1`–`S6`. Shown is mean $\pm$ standard deviation across 10 seeds. All methods use default hyperparameters and consistently reach 100% constraint satisfaction on `S1`–`S6`, except for IP on `S3` ($86 \pm 5\%$).

	PAL-LM	PAL-SQP	PAL-IP
`S1`	0.02 ± 0.02	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$
`S2`	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	0.02 ± 0.00
`S3`	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
`S4`	$\mathbf{0.01 \pm 0.00}$	$\mathbf{0.01 \pm 0.00}$	0.05 ± 0.02
`S5`	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
`S6`	0.08 ± 0.01	$\mathbf{0.07 \pm 0.01}$	0.21 ± 0.02

Each variant exposes one additional hyperparameter: the slack penalty for SQP (we use $\rho = 10^3$) and the barrier weight for the IP step (we use $\mu_0 = 0.01$).

Both are trained, just as with our LM repair step, with one detached repair step and iterate the same step at inference with a budget of 10 steps. The loss formulation and weight update rule remain unchanged from PAL. Beyond the above parameters ρ and μ, which were set to reasonable defaults but not tuned, we did not modify any other hyperparameters.

Table 6 reports the objective at convergence for the three repair steps. Feasibility is 1.00 ± 0.00 for every method on every benchmark, with the single exception of the IP step on `S3` (0.86 ± 0.05). We speculate that its barrier steps stall on the near-parallel, ill-conditioned rows of `S3` and exhaust the ten-step inference repair budget.

The SQP variant matches or slightly improves on LM on five of the six benchmarks, tying on `S2`, `S3`, and `S5` and improving on `S1`, `S4`, and `S6`, and never underperforms LM. This is consistent with SQP's coupled handling of the inequalities.

Overall, PAL accommodates all three repair operators without adaptation, which we interpret as PAL being agnostic to the specific repair step, provided the step is a linearized projection-type repair that is robust to ill-conditioned Jacobians.

E Additional Details on $f(\tilde{y})$ vs. $f(\hat{y})$ for Training Loss

In this appendix, we hypothesize about the causes of stalled training when $f(\tilde{y}^{(1)})$ is used in the PAL training loss. Intuitively, one would think that $f(\tilde{y})$ is the logical choice, since it is the objective after projection. One would want to take the change in objective by the projection into account. This is also the motivation for using $f(\tilde{y}^{(N)})$ in the baseline SOTA methods we considered in § 3.

However, we have found that using $f(\tilde{y})$ directly in the loss can have disastrous consequences on training convergence of the objective when projection is enabled in early training. In short, this is because using $f(\tilde{y})$ involves backpropagating its gradients through the projector. Here, we sketch out in rough terms that, because of how the projector is formulated for us (damped LM), the gradients $\frac{\partial f(\tilde{y})}{\partial \tilde{y}}$ get *projected* onto directions *tangent to the constraint manifold* $\mathcal{M}$, which, in early training[2], can be disastrous because the direction toward the global constrained optimum is, in general, not tangent to $\mathcal{M}$ at the repaired $\tilde{y}^{(1)}$ when (as is usual in early training) the unrepaired prediction $\hat{y}$ is bad (very far from constraint optimum).

We first derive how the projector formulation leads to this effect, demonstrate it on our simple example described in Appendix H, and then speculate why SOTA methods employing similar projectors do not report on this effect (as far as we are aware).

The Differential of One LM Step. We recall the definition of the constraint Jacobian $J = \partial c / \partial y$ (size $K \times n$, with K the number of active equality constraints and n the dimension of y), $M =$

[2]On the other hand, later in training, this can be desirable, since one would want objective gradients to reach N_θ only in directions tangent to $\mathcal{M}$ at $\tilde{y}$.

$JJ^\top + \lambda I_K$, and $\lambda = \delta \, \|c(\hat{y})\|^2$. The single LM step is

$$\tilde{y}^{(1)}(\hat{y}) \;=\; \hat{y} \;-\; J^\top M^{-1} c. \tag{5}$$

Applying the product rule gives

$$\frac{\partial \tilde{y}^{(1)}}{\partial \hat{y}} \;=\; I \;-\; J^\top M^{-1} J \;-\; \underbrace{(\partial J^\top / \partial \hat{y})\, M^{-1} c}_{\nabla_1} \;-\; \underbrace{J^\top (\partial M^{-1} / \partial \hat{y})\, c}_{\nabla_2}. \tag{6}$$

where both remainders ∇_1 and ∇_2 vanish as $\|c\| \to 0$, so that:

$$\frac{\partial \tilde{y}^{(1)}}{\partial \hat{y}} \;=\; \mathbf{P}_\lambda \;+\; O(\|c\|), \qquad \mathbf{P}_\lambda \;:=\; I - J^\top (JJ^\top + \lambda I)^{-1} J. \tag{7}$$

By the chain rule, $\partial f(\tilde{y}^{(1)})/\partial \hat{y} \approx \nabla f(\tilde{y}^{(1)})^\top \mathbf{P}_\lambda$, so the structure of $\mathbf{P}_\lambda$ determines the gradient-signal flow to the neural backbone N_θ when using $f(\tilde{y}^{(1)})$ in the training loss. Here, we just consider the case of using $f(\tilde{y}^{(1)})$, i.e., a single projection step.

Graphical Intuition. Take the simplest non-trivial case: one active equality constraint ($K = 1$) with output dimension $n = 2$. At $\tilde{y}^{(1)}$, let u be the unit tangent to the feasible manifold and v the unit normal (so $\nabla c \parallel v$). In the (u, v) basis, $J = (0, \sigma)$ with $\sigma = \|\nabla c\|$, hence $JJ^\top = \sigma^2$ and

$$\mathbf{P}_\lambda \;=\; \begin{pmatrix} 1 & 0 \\ 0 & \dfrac{\lambda}{\sigma^2 + \lambda} \end{pmatrix}.$$

Writing $\nabla f(\tilde{y}^{(1)}) = (f_u,\, f_v)^\top$ in the same basis,

$$\mathbf{P}_\lambda^\top \nabla f(\tilde{y}^{(1)}) \;=\; \begin{pmatrix} f_u \\ \dfrac{\lambda}{\sigma^2 + \lambda}\, f_v \end{pmatrix}.$$

The tangent component f_u is preserved. The normal component f_v is attenuated by $\lambda/(\sigma^2 + \lambda)$, which vanishes for small damping terms λ. (We use $\lambda = \delta\|c\|^2$ with $\delta = 10^{-3}$, which results in $\lambda \ll 1$). Using an undamped projection ($\lambda = 0$) step would entirely block f_v.

In contrast, when we use the *pre-projection* objective $f(\hat{y})$, then $\partial f(\hat{y})/\partial \hat{y} = \nabla f(\hat{y})^\top$ is unperturbed since it is not routed through $\mathbf{P}_\lambda$. Objective and penalty can compete in the same subspace and the adaptive weight $\bar{w}_c$ arbitrates that competition. When the objective is placed at $\tilde{y}^{(1)}$, it is band-limited to the tangent subspace and no longer participates in the direction that controls feasibility pressure on the backbone.

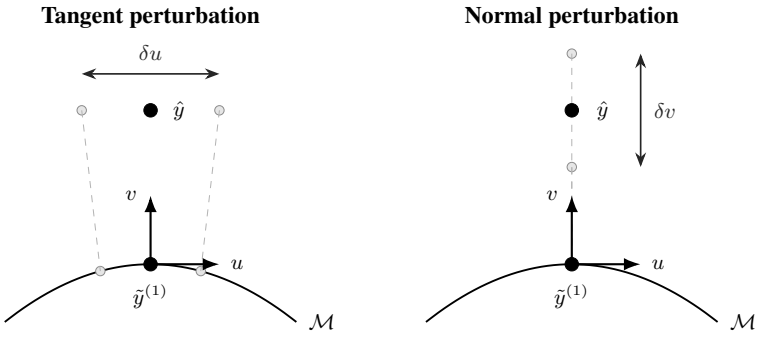


Figure 4: Geometric intuition. The projector eliminates gradient flow for $f(\tilde{y}^{(1)})$ along direction v (orthogonal to local constraint manifold $\mathcal{M}$) because, for an idealized projector, $\partial \tilde{y}^{(1)}/\partial \hat{y} \cdot \delta v = 0$. This leaves only gradients of $f(\tilde{y}^{(1)})$ which are *tangent* to the constraint manifold (direction u) to be backpropagated through the projection step into the neural backbone N_θ.

Empirical Demonstration. In Fig. 5, we show three variants of PAL trained on the toy problem described in Appendix H: (i) loss at $f(\hat{y})$ (PAL default), (ii) loss using $f(\tilde{y}^{(1)})$, and (iii) loss using $f(\hat{y})$ for the first half of training, switched to $f(\tilde{y}^{(1)})$ at half the epochs. Variant (i) converges normally. Variant (ii) is stuck near initialisation for the entire run: the gradient magnitude is roughly four orders of magnitude smaller than in (i). Variant (iii) shows that the failure mode is specific to early training: once the network has been pulled close to the feasible manifold under $f(\hat{y})$, the projector's normal-attenuation no longer harms training.

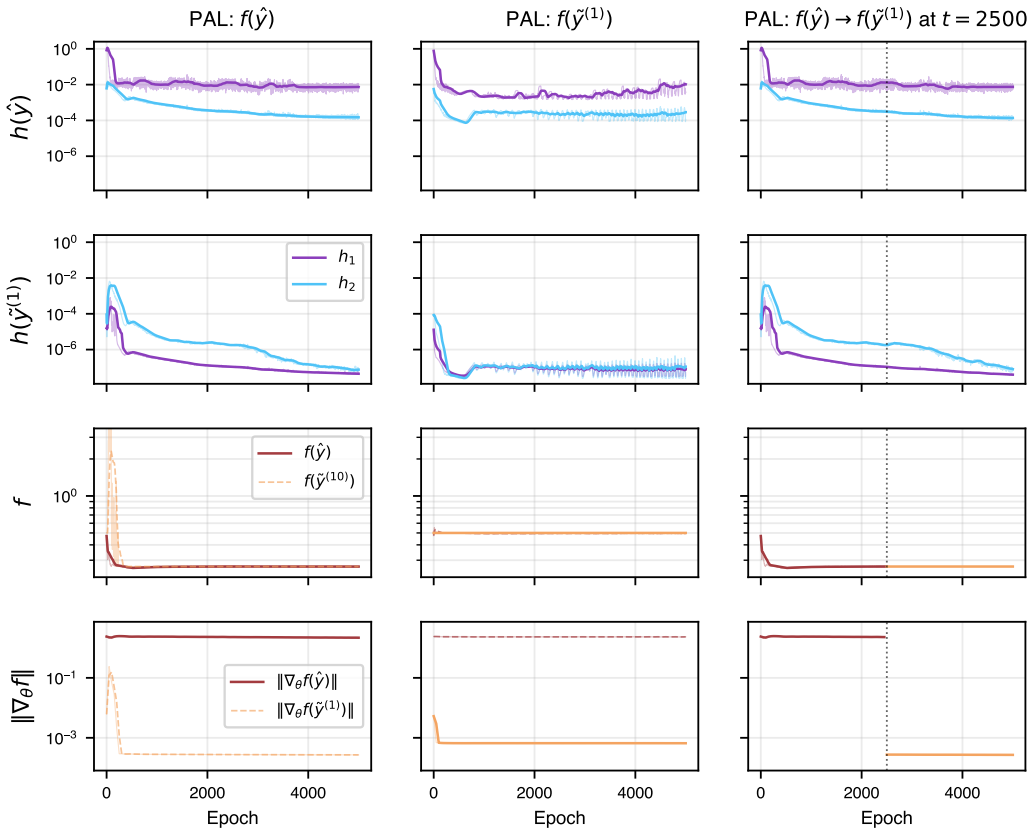


Figure 5: PAL loss-placement comparison on the toy problem of Appendix H. Rows from top: pre-projection residuals $h(\hat{y})$, post-projection residuals $h(\tilde{y}^{(1)})$, objective values f, and objective gradient magnitudes (computed on a fixed evaluation batch). Columns: $f(\hat{y})$ throughout (PAL default), $f(\tilde{y}^{(1)})$ throughout, and a midway switch from $f(\hat{y})$ to $f(\tilde{y}^{(1)})$ at $t = 2500$ (dotted vertical line). When using $f(\tilde{y}^{(1)})$ throughout, the objective never escapes the initialisation neighbourhood due to vanishing gradients normal to the manifold. The right column, having trained under $f(\hat{y})$ first, retains its objective and feasibility quality after the switch, demonstrating that the pathology is specific to early training.

Why do prior methods not report this? We hypothesize that the same gradient-routing pathology affects every projection-based method that places the objective at the post-repair iterate $\tilde{y}^{(N)}$, but that each SOTA method in the literature implicitly mitigates this issue with warm-up procedures: Both SnareNet (Chu et al., 2026) and ENFORCE (Lastrucci & Schweidtmann, 2025) defer hard projection to late training. SnareNet uses adaptive constraint relaxation, and ENFORCE activates its projection module only after a warm-up phase. We hypothesize that DC3 (Donti et al., 2021) and FSNet (Nguyen & Donti, 2025) are unaffected because they rely on gradient descent (FSNet wholly, DC3 for inequalities) where the differential of N small gradient-descent steps does not suffer from the issues described above.

Implications for PAL. In practice, to circumvent this issue with PAL, there are three approaches to deal with the effect described above:

1. *Pre-projection objective*. This is what we report on using throughout in this paper and, while not perfect in theory, is the most straightforward solution in practice. Since we enforce $\|\Delta y^{(1)}\| < \tau_d$ using our adaptive loss (see weight update in Eq. 2), we assume $f(\hat{y}) \approx f(\tilde{y}^{(1)})$ for $\tau_d \ll 1$ and simply use this.
2. *Warm-up*. One could consider using $f(\hat{y})$ in early training and switching to $f(\tilde{y}^{(1)})$ in late stages of training to combine the advantages of both. However, as Fig. 5 shows, there is virtually no advantage over simply using $f(\hat{y})$ throughout training when $\tau_d \ll 1$.
3. *Warm-up with Penalty Method*. Alternatively, one could simply skip projection in early training like many SOTA methods do, and use either $f(\hat{y})$ or $f(\tilde{y}^{(1)})$ later in training. For our experiments and algorithm presentation we omit this since it would add more complexity to our method. However, we would like to note that there would be some benefits to doing so also in terms of efficiency, since the additional compute of the single projection step would be eliminated during the warm-up phase. This could be especially relevant for $K \gg 1$, where assembly of the constraint Jacobian J is expensive.

F Levenberg–Marquardt Projector

We write $c \in \mathbb{R}^K$ for the constraint violation vector, with entries $|h_k(y)|$ for equalities and $\max(g_k(y), 0)$ for inequalities, and K the number of constraints.

Active Set. The active constraint vector $c_{\mathcal{A}}$ collects all violated constraints $c(y) > 0$. $J_{\mathcal{A}}(y) := \partial c_{\mathcal{A}}/\partial y$ is the Jacobian of the active constraint vector. The local feasible manifold of the active constraints is $\mathcal{M}_{\mathcal{A}} := \{y : c_{\mathcal{A}}(y) = 0\}$.

The LM step treats all active constraints as equalities and does not consider the inactive inequalities, i.e., they are deleted from the linear system. A mask is used for this using $c(\tilde{y}^{(i)})$ before every iterate. In effect, an inequality that a step overshoots enters the active set at the next iterate, and one that becomes satisfied leaves it. The active set can therefore switch between iterates at inference. In practice, we did not experience any issues with active-set switching. The convergence statement below assumes a fixed active set. The two alternative repair steps (SQP and Interior-Point) in Appendix D do not use this masking, since they both have native ways to account for non-violated inequalities (quadratic subproblem for SQP, barrier weight for IP).

LM Repair Step. Starting from $\tilde{y}^{(0)} = \hat{y}$, the LM projection applies:

$$\tilde{y}^{(i+1)} = \tilde{y}^{(i)} - J_{\mathcal{A}}(\tilde{y}^{(i)})^\top \left(J_{\mathcal{A}}(\tilde{y}^{(i)}) J_{\mathcal{A}}(\tilde{y}^{(i)})^\top + \lambda_i I \right)^{-1} c_{\mathcal{A}}(\tilde{y}^{(i)}), \tag{8}$$

with Yamashita–Fukushima (Yamashita & Fukushima, 2001) damping

$$\lambda_i = \delta \cdot \|c_{\mathcal{A}}(\tilde{y}^{(i)})\|^2,$$

where $\delta > 0$ is a constant (we use 10^{-3}). The projection is computed independently per batch element.

Each step of Eq. 8 equivalently solves

$$\Delta y^{(i)} = \arg\min_{\Delta y} \tfrac{1}{2} \left\| c_{\mathcal{A}}(\tilde{y}^{(i)}) + J_{\mathcal{A}}(\tilde{y}^{(i)})\, \Delta y \right\|^2 + \tfrac{\lambda_i}{2} \|\Delta y\|^2. \tag{9}$$

The Tikhonov term $\frac{\lambda_i}{2}\|\Delta y\|^2$ regularizes $J_{\mathcal{A}} J_{\mathcal{A}}^\top$ and bounds the step where the linearization is inaccurate. As $c_{\mathcal{A}} \to 0$ this term vanishes, equation 9 reduces to the minimum-norm solution of $J_{\mathcal{A}}\, \Delta y = -c_{\mathcal{A}}$, i.e., a first-order approximation to orthogonal projection onto $\mathcal{M}_{\mathcal{A}}$. The LM projector is insensitive to the damping hyperparameter δ over multiple orders of magnitude but removing damping entirely ($\delta = 0$ and $\lambda_{\min} = 0$) degrades the objective on some seeds (see Appendix B). On ill-conditioned constraints the undamped step fails, as we observe for the undamped completion of DC3 in Table 1.

Expanding $c_{\mathcal{A}}$ around $\tilde{y}^{(i)}$:

$$c_{\mathcal{A}}(\tilde{y}^{(i+1)}) = c_{\mathcal{A}}(\tilde{y}^{(i)}) + J_{\mathcal{A}}(\tilde{y}^{(i)})\big(\tilde{y}^{(i+1)} - \tilde{y}^{(i)}\big) + \mathcal{O}(\|\Delta y^{(i)}\|^2).$$

With damping the LM step does not drive the linearized residual to zero, so $R_{i+1} = c_{\mathcal{A}}(\tilde{y}^{(i+1)})$ collects two contributions: a damping residual and the higher-order Taylor remainder $\mathcal{O}(\|\Delta y^{(i)}\|^2)$ under $c_{\mathcal{A}} \in C^1$ with locally Lipschitz Jacobian.

For the idealized fixed-active-set smooth case (Assumption 1), Yamashita & Fukushima (2001) establish local quadratic convergence of Eq. 8 to the active feasible set $\mathcal{M}_{\mathcal{A}}$ from a *sufficiently close initialization*, which we use as motivation for our adaptive weight update rule.

F.1 Convergence

We fix a neighborhood $U \subset \mathbb{R}^d$ on which the active set A is constant, and let $c_{\mathcal{A}} : U \to \mathbb{R}^{K_{\mathcal{A}}}$ denote the corresponding smooth active residual map, with Jacobian $J_{\mathcal{A}}(y) := \nabla c_{\mathcal{A}}(y)$. The local active feasible set is $\mathcal{M}_{\mathcal{A}} := \{y \in U : c_{\mathcal{A}}(y) = 0\}$.

Assumption 1 (Local active-set regularity)**.** *There exists a point $y^\star \in \mathcal{M}_{\mathcal{A}}$ and a neighborhood U of $y^\star$ such that:*

1. *the active set A is fixed on U.*
2. *$c_{\mathcal{A}}$ is continuously differentiable on U, and $J_{\mathcal{A}}$ is Lipschitz continuous on U.*
3. *$c_{\mathcal{A}}$ satisfies a local error bound on U: there exists $\kappa > 0$ such that*

$$\operatorname{dist}(y, \mathcal{M}_{\mathcal{A}}) \le \kappa \|c_{\mathcal{A}}(y)\| \qquad \forall y \in U.$$

The error bound in (3) is strictly weaker than requiring $J_{\mathcal{A}}$ to have full row rank on $\mathcal{M}_{\mathcal{A}}$, and accommodates redundant or rank-deficient constraint systems.

Under Assumption 1, Theorem 2.1 of Yamashita & Fukushima (2001) applies to the idealized fixed-active-set iteration: for all $\tilde{y}^{(0)}$ sufficiently close to $\mathcal{M}_{\mathcal{A}}$, the iterates of Eq. 8 converge quadratically to $\mathcal{M}_{\mathcal{A}}$. In particular, there exist $C, \rho > 0$ such that whenever $\operatorname{dist}(\tilde{y}^{(i)}, \mathcal{M}_{\mathcal{A}}) \le \rho$,

$$\operatorname{dist}(\tilde{y}^{(i+1)}, \mathcal{M}_{\mathcal{A}}) \le C \operatorname{dist}(\tilde{y}^{(i)}, \mathcal{M}_{\mathcal{A}})^2.$$

Combining with the error bound and the local Lipschitz property of $c_{\mathcal{A}}$, this yields $r_{i+1} \le M r_i^2$ with $M := LC\kappa^2$, where $r_i := \|c_{\mathcal{A}}(\tilde{y}^{(i)})\|$ (Yamashita & Fukushima, 2001). This applies to the smooth fixed-active-set map $c_{\mathcal{A}}$, not automatically to the deployed practical projector with active-mask updates and numerical damping floor $\lambda_{\min}$.

Practical Proxy for the Local Basin. The radius ρ is problem-specific and not directly observable. Accordingly, we do not claim that $R_{1,k} < \tau_c$ is sufficient for the Yamashita–Fukushima basin condition for the idealized fixed-active-set iteration. Instead, we use $R_{1,k}$ only as a practical heuristic for local convergence. If a single linearized repair step from $\hat{y}$ drives all constraint residuals below τ_c, then the local linear model is accurate at $\hat{y}$, and the projector is likely operating in a regime of reliable local convergence. This is the signal used by PAL's weight rule (Eq. 2), which treats constraint k as easy to repair when its single-step residual falls below τ_c. Empirically, for sufficiently small problem-dependent choices of τ_c (typically around 10^{-4}), this proxy has aligned well with stable projector behavior across our experiments up to extremely non-linear constraints (high curvature) as shown in § 5.3.

G Inference Precision

Here, we use checkpoints from the paper on the six synthetic benchmarks `S1`–`S6` which were trained with $\tau_c = \tau_d = 10^{-4}$. We show inference with iterates of the repair loop for $N \in \{1, 2, 5, 10, 100\}$ steps and record the maximum constraint violation over the test set of 64 queries per seed, averaged over 10 seeds. We report `float64` and `float32` implementation variants.

Table 7 shows that for both `float64` and `float32`, every benchmark plateaus at the numerical precision floor from $N \approx 2$ to 5 onward, except for `float64` on benchmark `S5`. We believe this is due to `S5`'s overdetermined active set. The violation decreases monotonically in N on every benchmark.

Overall, this illustrates that the first-step residual target $\tau_c = 10^{-4}$ and the actual precision that can be reached for $N > 1$ are not to be confused, and that PAL can consistently reach arbitrary precision.

Table 7: Maximum constraint violation over the test set against the number of inference repair steps N, over 10 seeds, for the trained PAL models from the paper on the synthetic benchmarks `S1`–`S6`.

	N=1	N=2	N=5	N=10	N=100
`float64`					
`S1` (sphere track)	$8.48 \cdot 10^{-4}$	$2.46 \cdot 10^{-7}$	$7.55 \cdot 10^{-16}$	$7.55 \cdot 10^{-16}$	$7.55 \cdot 10^{-16}$
`S2` (active-set switch)	$1.36 \cdot 10^{-4}$	$3.41 \cdot 10^{-6}$	$2.22 \cdot 10^{-16}$	$2.22 \cdot 10^{-16}$	$2.22 \cdot 10^{-16}$
`S3` (ill-cond. tube)	$8.42 \cdot 10^{-6}$	$1.03 \cdot 10^{-7}$	$3.54 \cdot 10^{-15}$	$2.22 \cdot 10^{-17}$	$2.01 \cdot 10^{-17}$
`S4` (qv coupling)	$1.01 \cdot 10^{-2}$	$1.26 \cdot 10^{-5}$	$8.44 \cdot 10^{-16}$	$9.10 \cdot 10^{-16}$	$8.22 \cdot 10^{-16}$
`S5` (overdetermined)	$6.77 \cdot 10^{-3}$	$4.98 \cdot 10^{-4}$	$6.33 \cdot 10^{-8}$	$8.81 \cdot 10^{-9}$	$8.83 \cdot 10^{-11}$
`S6` (redundant ineq.)	$2.16 \cdot 10^{-3}$	$1.45 \cdot 10^{-5}$	$1.67 \cdot 10^{-16}$	$1.39 \cdot 10^{-16}$	$1.33 \cdot 10^{-16}$
`float32`					
`S1` (sphere track)	$8.48 \cdot 10^{-4}$	$5.81 \cdot 10^{-7}$	$3.58 \cdot 10^{-7}$	$3.58 \cdot 10^{-7}$	$3.58 \cdot 10^{-7}$
`S2` (active-set switch)	$1.36 \cdot 10^{-4}$	$3.54 \cdot 10^{-6}$	$1.43 \cdot 10^{-7}$	$1.43 \cdot 10^{-7}$	$1.43 \cdot 10^{-7}$
`S3` (ill-cond. tube)	$8.49 \cdot 10^{-6}$	$1.47 \cdot 10^{-7}$	$3.13 \cdot 10^{-8}$	$3.05 \cdot 10^{-8}$	$2.76 \cdot 10^{-8}$
`S4` (qv coupling)	$1.01 \cdot 10^{-2}$	$1.37 \cdot 10^{-5}$	$3.96 \cdot 10^{-7}$	$4.49 \cdot 10^{-7}$	$4.15 \cdot 10^{-7}$
`S5` (overdetermined)	$6.74 \cdot 10^{-3}$	$5.74 \cdot 10^{-4}$	$2.80 \cdot 10^{-7}$	$2.62 \cdot 10^{-7}$	$2.38 \cdot 10^{-7}$
`S6` (redundant ineq.)	$2.16 \cdot 10^{-3}$	$1.50 \cdot 10^{-5}$	$1.24 \cdot 10^{-7}$	$1.19 \cdot 10^{-7}$	$1.19 \cdot 10^{-7}$

H Residual Intuition on a Toy Problem

Fig. 2 compares three constraint-handling strategies on a single 2D problem with one stiff linear equality and one mildly nonlinear equality:

$$\min_{y} \; (y_1 - 0.5)^2 + (y_2 - 0.5)^2 \quad \text{s.t.} \quad \begin{aligned} h_1(y) &= 10\, y_1 = 0, \\ h_2(y) &= 0.01 \sin(5\, y_2) + 0.1\, y_1 = 0. \end{aligned}$$

The objective pulls the prediction toward $(0.5, 0.5)$. h_1 is stiff and linear (well-suited to a single LM step), while h_2 is mildly nonlinear in y_2 and weakly couples back to y_1. The neural backbone is a 3-layer MLP of width 128 with ReLU activations, trained with Adam at learning rate 10^{-4} for 5000 epochs and batch size 100. Conditions z are sampled from $\mathcal{N}(0, I_2)$ and supplied as input but do not enter the problem.

All three methods use the *same* LM projector with identical hyperparameters (damping $\delta = 10^{-3}$, floor 10^{-6}) both during training (where invoked) and at evaluation. They differ only in (i) how often the projector is invoked during training, and (ii) the training loss:

- **ALM+Bolt-On.** No projection during training. MDMM-style dual ascent (Platt & Barr, 1988) with fixed quadratic penalty: $\mathcal{L} = f + \lambda^\top h + \frac{\rho}{2}\|h\|^2$, $\lambda \leftarrow \lambda + \eta_\lambda\, h$, with $\eta_\lambda = 10^{-2}$ and $\rho = 1$. The projector is bolted on only at evaluation. (Basir & Senocak's APU (Basir & Senocak, 2023) was unstable on this toy problem, so we substitute MDMM as a closely related fixed-penalty variant.)
- **Soft Loss.** Ten chained LM projection steps in training with live gradient through the displacement chain, trained with $\mathcal{L} = f + \lambda_d\, \|\hat{y} - \tilde{y}^{(10)}\|_2 + \lambda_c\, \|h(\tilde{y}^{(10)})\|_2$ at $\lambda_d = 0.5$, $\lambda_c = 1.0$.
- **PAL.** A single LM projection step in training, with the gated log-decade weight update of Eq. 2: $\gamma = 10^{-2}$, $D_{\max} = 1$, $\tau_d = 10^{-1}$.

I SYNTHETIC BENCHMARK DETAILS

We provide formal definitions for the conditional synthetic benchmarks. All follow the convention $h(y, z) = 0$ for equalities and $g(y, z) \leq 0$ for inequalities.

Five of the six benchmarks (`S1`–`S4`, `S6`) share a *degenerate-objective* construction: the design variable y is partitioned into a *head* block on which the objective depends and a *tail* block that does not appear in the objective but is coupled to the head through one or more constraints. Projecting an arbitrary $\hat{y}$ onto the constraint set generally moves the head coordinates away from their optimum, so the backbone network N_θ must drive the constraint residual toward zero on its own, since projection alone is not enough. `S5` is an overdetermined consistent equality system whose feasible set is finite per z, isolating the projection geometry from objective effects.

`S1` (Sphere constraint). Design variable $y \in \mathbb{R}^4$, $K = 1$, condition $z \in [0.5, 2]$. One equality, no inequalities.

$$\min_y \big(1 - (y_1 - s_1(z))\big)^2 + 100\big((y_2 - s_2(z)) - (y_1 - s_1(z))^2\big)^2 \quad \text{s.t.} \quad \textstyle\sum_{i=1}^4 y_i^2 = z^2,$$

where the shift $s(z)$ is chosen so the global minimizer is $y^\star(z) = (z/\sqrt{2}, z/\sqrt{2}, 0, 0)$. The objective depends only on (y_1, y_2), and the sphere projection scales all four coordinates uniformly, so any tail offset displaces the head off-optimum after projection.

`S2` (Probability simplex). Design variable $y \in \mathbb{R}^{10}$, $K = 11$, condition $z \in [-1, 1]^2$. One equality and ten non-negativity inequalities.

$$\min_y \big(s(z) - y_1\big)^2 + 10 \sum_{i=1}^4 (y_{i+1} - y_i^2)^2 \quad \text{s.t.} \quad \textstyle\sum_{i=1}^{10} y_i = 1, \quad y_i \geq 0,$$

with $s(z) = 0.4 + 0.05\, z_1 + 0.02\, z_2$. Only the head $(y_1, \ldots, y_5)$ enters the chained-Rosenbrock objective, and $\sum_{i \leq 5} y_i^\star \approx 0.586$, so the simplex equality forces nontrivial mass onto the tail $(y_6, \ldots, y_{10})$, which does not enter the objective.

`S3` (Ill-conditioned head with mixing equality). Design variable $y \in \mathbb{R}^{20}$, $K = 6$, condition $z \sim \mathrm{LogU}(0.1, 1)$. One mixing equality, one nonlinear tube inequality, four near-parallel linear inequalities.

$$\min_y \ \tfrac{1}{2} \sum_{i=1}^{10} z^{2(i-1)/9} \big(y_i - t_i(z)\big)^2 \quad \text{s.t.} \quad \begin{aligned} &a_0^\top y = a_0^\top t(z), \quad \|y_{11:20}\|^2 \leq z^4, \\ &a_j^\top y \leq a_j^\top t(z) + 0.2 \quad (j = 1, \ldots, 4), \end{aligned}$$

with target $t_i(z) = z$ for $i \leq 10$ and $t_i(z) = z^2/\sqrt{10}$ for $i > 10$. The head Hessian is diagonal with condition number z^{-2}, ranging up to 100 at $z = 0.1$. The mixing equality couples the head $y_{1:10}$ to the tail $y_{11:20}$ (which does not enter the objective), so any tail movement to satisfy the equality feeds back into the head through the projection step.

`S4` (Quadratic objective with sinusoidal coupling). Design variable $y = (u, v) \in \mathbb{R}^{15} \times \mathbb{R}^{15}$, $K = 26$, condition $z \in [0.5, 1.5]^5$. Twenty-five equalities and one inequality.

$$\min_y \ \tfrac{1}{2}\big(v - v^\star(z)\big)^\top W_v \big(v - v^\star(z)\big) \quad \text{s.t.} \quad \begin{aligned} &A\, u = \tilde{A}\, z, \quad v_j = \sin(u_j) + (Bz)_j, \\ &\|v\|^2 \leq \|v^\star(z)\|^2 + 1, \end{aligned}$$

with $A \in \mathbb{R}^{10 \times 15}$ rank 10 and W_v SPD with $\mathrm{cond}(W_v) \approx 5$. The block u does not enter the objective, but the main equality is underdetermined in u, so projecting an arbitrary $\hat{u}$ onto the constraint produces a point that generically differs from $u^\star$, and the sinusoidal coupling then forces v off its quadratic optimum.

`S5` (Overdetermined system). Design variable $y \in \mathbb{R}^4$, $K = 5$, condition $z \in [0.5, 1.5]^5$. Five equalities ($m > n$), no inequalities, mutually consistent at a $y^\star(z) \in \mathbb{R}^4$ parameterized by z.

$$\min_y \ \tfrac{1}{2}\big(y - y^\star(z)\big)^\top P \big(y - y^\star(z)\big) \quad \text{s.t.} \quad \begin{aligned} &y_1^2 + y_2^2 = z_1^2, \quad y_3^2 + y_4^2 = z_2^2, \\ &y_1 y_3 = \alpha(z), \quad y_2 y_4 = \beta(z), \\ &y_1 y_4 - y_2 y_3 = \delta(z), \end{aligned}$$

where $\alpha(z) = y_1^\star y_3^\star$, $\beta(z) = y_2^\star y_4^\star$, and $\delta(z) = y_1^\star y_4^\star - y_2^\star y_3^\star$ are the corresponding bilinear forms of $y^\star(z)$. Because $m > n$, the implicit-function partition required by DC3 does not apply. The feasible set is $\{\pm y^\star(z)\}$, so any successful projection lands at one of these two points regardless of $\hat{y}$.

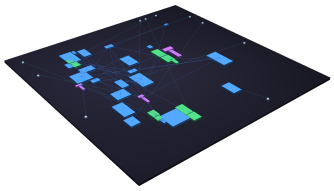
`S6` (Coupling equalities with redundant inequalities). Design variable $y \in \mathbb{R}^{50}$, $K = 90$, condition $z \in [0.5, 1.5]^{20}$. Fifteen coupling equalities and 75 inequalities (25 distinct + 50 near-redundant copies).

$$\min_y \ \tfrac{1}{2}\sum_{i=16}^{50}\big(y_i - t_i(z)\big)^2 \quad \text{s.t.} \quad \begin{aligned} & y_i - \tfrac{1}{2}y_{i+15} = z_{(i \bmod 20)} - \tfrac{1}{2}z_{((i+15) \bmod 20)}, \ \ i = 1, \dots, 15, \\ & a_j^\top y \le a_j^\top t(z) + 0.2, \ \ j = 1, \dots, 25, \end{aligned}$$

The objective depends only on the tail $y_{16:50}$, and the head $y_{1:15}$ is pinned to the tail via the coupling equalities. The redundant inequality block stresses projector behavior (rank-deficient active sets).

J Chip Placement Case Study

Here, we show one additional application where the constraint and objective evaluation is relatively cheap, so that methods with unrolled repair in training are memory-feasible. This application is a macro-placement problem inspired by Intel's FloorSet (Mallappa et al., 2024). The task is to place $B = 30$ rectangular blocks on a 200×200 canvas to minimise a wirelength proxy (a weighted sum of pairwise Manhattan distances between connected block centres and boundary pins) plus a soft bounding-box area penalty (LSE-smoothed). A single equality constraint requires the total overlap volume across all $B(B-1)/2$ block pairs to be zero. The design vector $y \in \mathbb{R}^{3B}$ ($3B = 90$ variables) encodes block centres (x_i, y_i) and raw widths s_i. Widths decode as $w_i = \mathrm{softplus}(s_i) + 0.5$, and heights follow from a fixed per-instance target area, keeping all boxes non-degenerate and area-preserving. We want to point out that this benchmark is a lightweight component placement task and a strongly simplified FloorSet derivative. The full FloorSet suite additionally considers grouping, multi-instantiation, and pre-placement constraints.



	Obj. $\times 10^3 \downarrow$	Feas. (%) $\uparrow$	Wall-Clock $\downarrow$	n
PAL	$\mathbf{1.16 \pm 0.04}$	$\mathbf{100 \pm 0}$	1m18s±6s	10
ALM+Bolt-On	1.19 ± 0.05	80 ± 42	**40**s±**2**s	10
FSNet	5.27 ± 0.31	0 ± 0	1h36m±6m	5
SnareNet	1.35 ± 0.02	0 ± 0	1h26m±1m	5

Figure 6: Candidate chip placement for `E4` (left) and full results (right). PAL and ALM+Bolt-On run at 10 seeds, FSNet and SnareNet at 5 seeds. All methods use *canonical default hyperparameters without per-method tuning*.

Results. PAL reaches 100% feasibility while FSNet and SnareNet reach 0% feasibility at substantially higher wall-clock time than PAL due to the compute required by the unrolled repair. We show ALM, PAL, and SnareNet with 2000 training epochs each, while FSNet is shown with only 100 training epochs in order to keep wall-clock times manageable. We use canonical default hyperparameters for each method, i.e., no per-method tuning. Despite the problem having only one inequality constraint, both SnareNet and FSNet reach 0% feasibility. We did not investigate in further detail why this is the case, but believe both FSNet and SnareNet could perform better on this task if their hyperparameters were tuned extensively. However, orthogonal to this, we want to point out that PAL is *significantly* faster to train and achieves 100% feasibility despite not being tuned for the problem at hand. Computationally, this benchmark is very well suited to PAL since it only has a single constraint ($K = 1$), meaning that the constraint Jacobian J is cheap to build. ALM+Bolt-On reaches 100% feasibility on 8 of 10 seeds, but the remaining 2 seeds carry very high constraint violations, which is what drives the mean to $80\% \pm 42\%$ in Fig. 6. This experiment was run in parallel (across seeds and methods) on a 288-core ARM CPU node. Cross-method ratios for wall-clock time are reliable, but absolute wall-clock numbers are hardware- and contention-specific.

K Engineering Benchmark Details

K.1 Unmanned Aircraft

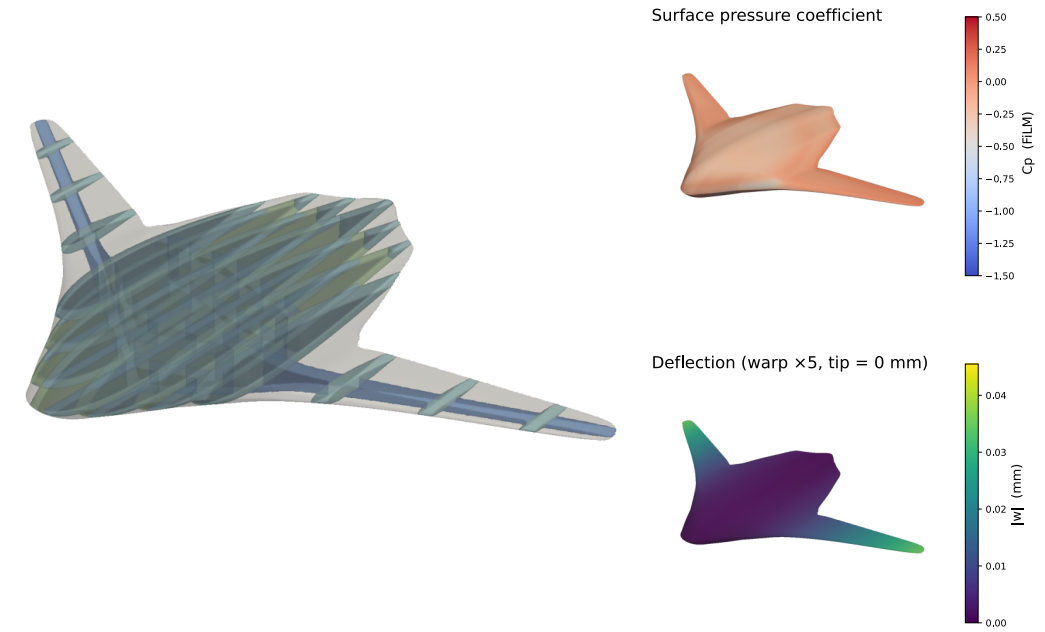


Figure 7: `E1` unmanned aircraft benchmark, candidate design. **Left:** outer mould-line geometry from the analytical Bézier representation, with the decoded internal wingbox (skin, spars, ribs) overlaid in blue/green. **Top right:** surface pressure coefficient c_p predicted by the FiLM-modulated BlendedNet surrogate $\Phi^{\theta}_{\text{cp}}$ (Sung et al., 2025) at iso-sliced surface points. **Bottom right:** structural tip deflection $|w(y)|$ from beam integration of the pressure loads through the predicted section properties, warp-amplified $5\times$ for visualisation. The tip-deflection constraint reads w_{tip} at $y = b/2$.

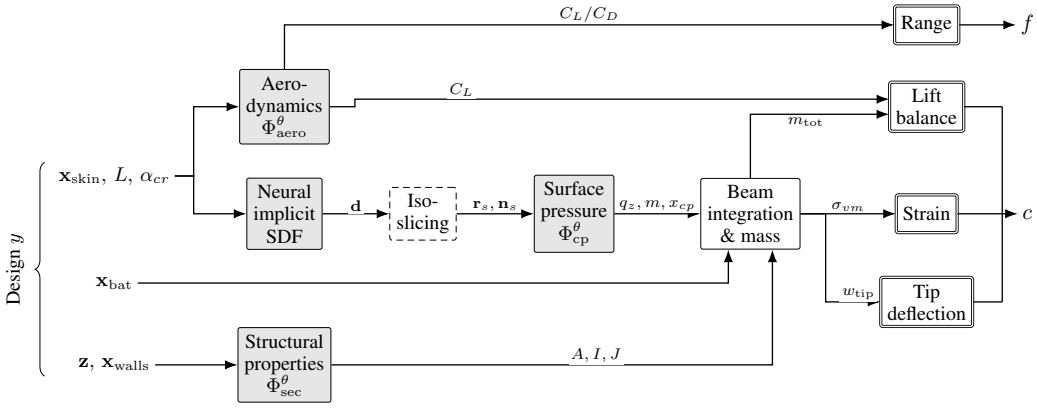


Figure 8: `E1` evaluator Φ, from the design vector y to the objective f and constraints c. Dashed: geometry processing. Plain: analytical. Grey: neural surrogate. Hatched: generative model. Double border: constraint or objective.

We now describe the benchmark `E1` in detail. This application aims to maximize the range on an unmanned Blended-Wing-Body (BWB) aircraft under structural constraints. Here, we mostly describe the evaluator Φ, which is made up of multiple *pre-trained* generative models and neural surrogates, specialized for various aspects of aircraft design. Fig. 8 shows the arrangement of these submodels. Fig. 7 shows a candidate design (geometry, surface pressure, tip deflection). The data and trained models are available in the supplied code repository.

K.1.1 GEOMETRY

The geometry for this application is based on BlendedNet (Sung et al., 2025). In order to achieve differentiability for our evaluator, we use geometry representations based on implicits (SDFs). We employ two different representations: a neural implicit SDF net, and an analytical implicit shape representation.

Neural Implicit BWB Geometry. First, we use a neural implicit representation for fast, differentiable signed-distance queries used inside the optimization loop during training (iso-slicing, surface sampling of the FiLM model, geometry-dependent constraints). We use a conditional MLP with GELU activations and Fourier positional encoding on the spatial input (Tancik et al., 2020; Mildenhall et al., 2020). This maps spatial points $\mathbf{r} \in \mathbb{R}^3$ and the 9-D shape conditioner $\mathbf{x}_{\text{skin}} = (B_1, B_2, B_3, C_2, C_3, C_4, S_1, S_2, S_3)$, which describe body widths, chords, sweeps, as described by Sung et al. (2025). We use an additional parameter L for isotropic scaling of the geometry. For each query (in practice, this is highly parallelized), a signed distance $\mathbf{d} \in \mathbb{R}^1$ (the distance to the surface) is returned. We use symmetry as a prior for added computational efficiency, such that $y \to |y|$ is enforced inside the network. The neural implicit representation is trained on the geometry dataset (meshes) provided in BlendedNet (Sung et al., 2025).

Analytical Bézier SDF. Similarly, this maps $\mathbf{r}, \mathbf{x}_{\text{skin}} \mapsto \mathbf{d}$ and serves as a high-fidelity reference geometry used for visualization (rendered with PyVista (Sullivan & Kaszynski, 2019)). This representation is much more computationally expensive because each query rebuilds Bézier patches and six CSG operations, and is therefore not used live in training. The 9 design parameters drive three spanwise wing panels, each a bicubic Bézier surface patch ($4\times4\times3$ control net) per upper/lower surface, anchored at four spanwise stations with fitted airfoil control points. For tangent handling between these patches at the trailing and leading edges (to match the planform described by Sung et al. (2025)), we use AdvLink-style in/out sweep tangents per station that replicate the OpenVSP (McDonald & Gloudemans, 2022) tangent conventions. We use per-panel surface SDF $\cap$ y-slab (smooth) $\cap$ swept TE half-space (sharp) to make a smooth union over panels, then mirror this across the xz plane to make a smooth union of the two halves into the full aircraft. To find the bicubic Bézier control points, we performed regression on the airfoil described by Sung et al. (2025). Since the splines only use 4 control points, we cannot model the airfoil exactly, resulting in a small RMSE of ≈ 2.7 mm/m (body) and ≈ 2.0 mm/m (tip).

Coordinate Frame. We denote x-direction as the direction of travel for the BWB, y-direction spanwise, and z-direction up/down (thickness-wise).

K.1.2 AERODYNAMICS

We use a simple MLP to model global aerodynamic parameters given a geometry ($\mathbf{x}_{\text{skin}}$) and flight state, i.e., no geometry processing is used for these predictions (instead, geometry is needed for aero-structural coupling, more on this below). We model lift, drag, and pitching-moment coefficients (C_L, C_D, C_M) with a simple MLP $\Phi^{\theta}_{\text{aero}}$ that feeds a lift-balance constraint and a Breguet-range objective. Inputs are the 9-D shape conditioner $\mathbf{x}_{\text{skin}} \in [-1, 1]^9$ from Sung et al. (2025). This is the same geometry parametrization the implicit representation above consumes. Additionally, we use the 4-D flight state $(\mathrm{Ma},\ h,\ \alpha_{cr}^{\deg},\ \log_{10} L)$, where Ma is derived from (V, h) via a standard-atmosphere helper, α_{cr} is converted from radians, and the characteristic length L is log-transformed (length scales span roughly one decade across training). For training data, we use the cases provided by Sung et al. (2025). Since this is a pure MLP with smooth activations, we can differentiate (C_L, C_D, C_M) w.r.t. $(\mathrm{Ma},\ h,\ \alpha_{cr}^{\deg},\ L)$.

K.1.3 SURFACE PRESSURE

We use the model denoted as $\Phi^{\theta}_{\text{cp}}$ to determine surface pressure on the BWB (top-right panel of Fig. 7), which is then used to determine wing deformation and strain in conjunction with our structural model (see below). This model is an exact training replica of the FiLM-modulated MLP from BlendedNet (Sung et al., 2025). The model is queried by the iso-slice mechanism (described below) with surface points $\mathbf{r}_s$ and surface normals $\mathbf{n}_s$ and a condition vector containing α_{cr}, the flow state, and shape parameters. The model returns the surface pressure coefficient c_p together with

chordwise and vertical shear coefficients $(c_{f,x}, c_{f,z})$. These are combined into the per-point surface force density $\mathbf{f} = -c_p\,\mathbf{n} + (c_{f,x}, 0, c_{f,z})$ before chordwise quadrature, so both pressure and skin friction enter the structural load path. Global drag is still taken from $\Phi^{\theta}_{\text{aero}}$ (see above), not integrated from the FiLM shear field.

K.1.4 ISO-SLICING

In order to couple the surface pressure model (FiLM BlendedNet) and our structural model, we employ a deterministic geometry extraction algorithm that queries the neural SDF using a quadtree-based marching-squares level-set extraction to find a discrete set of surface query points $\mathbf{r}_s$ with outward normals $\mathbf{n}$ and arc-length integration weights, suitable for chordwise quadrature of the FiLM predictions. At each spanwise station y_i, the BWB geometry is sliced with this iso-contour primitive `prog.isocontour(plane='xz', stations=y_unit, normal_mode='3d')`, which returns 2D polylines on the chordwise plane plus matching 3D outward surface normals and per-segment arc-length weights. Default quadtree settings use base resolution 8 with 5 levels of refinement, so the SDF is queried repeatedly per slice. After extraction, each chordwise polyline (x, z) is lifted to 3D by inserting the unit-frame spanwise station value as y. The station coordinate is detached from the gradient graph since it is a sampling location, not a design variable, such that the gradient graph only routes to $\mathbf{x}_{\text{skin}}, L, \alpha_{cr}$. At queried stations beyond the local span the iso-contour returns no polyline. The corresponding station's load contributions are zeroed and stay differentiable. Downstream, the outputs of the iso-slicing procedure $(\mathbf{r}_s, \mathbf{n}, w)$ are fed into the surface pressure model $\Phi^{\theta}_{\text{cp}}$ to determine c_p, which is then integrated chordwise with the arc-length weights into per-station aerodynamic loads. These are shear load, moment, and center of pressure (q_z, m, x_{cp}).

K.1.5 STRUCTURAL MODEL

The structural model of the BWB consists of a generative model, $\Phi^{\theta}_{\text{dec}}$ (Variational Autoencoder (VAE) (Kingma & Welling, 2014) decoder) to encode valid structural internals layouts conditional on the aircraft skin geometry $\mathbf{x}_{\text{skin}}$ and a surrogate model $\Phi^{\theta}_{\text{sec}}$ (MLP) to predict structural properties directly from the latent of this encoder/decoder. Data to train the VAE was obtained through rejection sampling of valid/invalid internal layout designs.

Architecture. The VAE encoder (used in training only) maps $(\mathbf{x}_{\text{struct}}, \mathbf{x}_{\text{skin}}) \to \mathbf{z} \in \mathbb{R}^d$. The decoder maps $\Phi^{\theta}_{\text{dec}}$: $(\mathbf{z}, \mathbf{x}_{\text{skin}}) \to \mathbf{x}_{\text{struct}} \in \mathbb{R}^9$, the reconstructed raw wingbox layout (skin, spar, rib geometry), shown overlaid in the left panel of Fig. 7. The decoder is used for training and visualization only. In the evaluator, the latent $\mathbf{z}$ is part of the design vector and feeds the structural properties predictor directly. As alluded to, the structural properties predictor maps $\Phi^{\theta}_{\text{sec}}$: $(\mathbf{z}, \mathbf{x}_{\text{skin}}, \mathbf{x}_{\text{walls}}, y)$ to structural properties, where $\mathbf{x}_{\text{walls}}$ is the design vector of (uniform) wall thickness of the BWB skin, front spar, and rear spar. For any given design at each training epoch, the structural model is used to evaluate structural properties per station. Additionally, we use a dedicated head to predict the overall volume of the inner ribs, which we assume do not contribute to the global stiffness constraints we evaluate, which is a conservative assumption. This dedicated head maps $(\mathbf{z}, \mathbf{x}_{\text{skin}}, t_{\text{rib}}) \to V_{\text{ribs}}$, where the rib volume V_{ribs} is a single scalar. This head is not used in the evaluator, and rib mass is not included in the mass aggregation below. All four components of the structural model architecture are plain GELU MLPs. We use latent dimension $\dim(\mathbf{z}) = 16$. The model explicitly does not handle the thickness of the ribs and spars, since skin thickness does not affect the geometric validity of structural internals and would have made rejection sampling much more expensive (curse of dimensionality) without additional benefit. Instead, these are concatenated after the decoder. For efficiency, the predictor is called once flattened over $B \cdot N_y$ stations: $\mathbf{z}$ and $\mathbf{x}_{\text{skin}}$ are broadcast across N_y Sobol-sampled spanwise stations and the predictor handles these in a single batched forward. Training data is based on quadtree-based marching-squares material point assignment and computing geometric section properties on the resulting cross-sections. The surrogate's ground truth is therefore based on SDF evaluation and beam-section integrals over the built-up wingbox, not FEA. The 11-D output covers cross-section area A (used to estimate structural mass via spanwise integration), centroid offsets (u_{cg}, v_{cg}), principal bending stiffnesses I_{uu} and I_{vv}, the per-axis first moments $Q_u^{\max}, Q_v^{\max}$ (used in the transverse-shear formula below, with $Q_{\max} = \max(Q_u^{\max}, Q_v^{\max})$), and torsional stiffness J. The shear centre is not surrogate-predicted. We assume $x_{sc} \equiv x_{cg}$ under doubly-symmetric thin-walled idealisation. As mentioned, we assume that the ribs do not contribute to stiffness and only compute their volume for mass estimation. Thus, stiffness stems from the main spar, as well as the skin. The

skin is extremely thin but its contributions to torsional and bending stiffness properties cannot be neglected, so in data generation we use very fine resolutions ($\sim 5\times$ skin thickness) when performing quadtree-based marching-squares in order to properly resolve the skin.

K.1.6 HELPER FUNCTIONS

We use deterministic differentiable helper functions for planform and atmospheric conditions. The planform evaluation $(S_{\text{ref}}, b/2)$ is a closed-form trapezoid integral over the four-station BWB chord distribution. The atmosphere helper is used to determine $\rho, q_\infty, \mathrm{Ma}, \mathrm{Re}$ using the ISA troposphere model (lapse rate to 11 km), in torch-native and autograd-safe formulations so all four quantities differentiate through (h, V) (altitude and velocity).

K.1.7 BEAM QUADRATURE, MASS, AND CONSTRAINTS

Spanwise Sampling. We use Sobol-quasi-random stations on $[0, b/2]$ per station query in each training epoch. Stations are sorted ascending and gradients flow through them via $b/2$ (the random draws themselves are detached).

Beam Integration. With the models described above, we obtain per-station distributed loads q_z, m, x_{cp} from Φ^θ_{cp} and section properties $A, I_{uu}, I_{vv}, J, u_{cg}, v_{cg}$ from the structural prediction Φ^θ_{sec}. First, we integrate the loads:

$$V(y) = \int_y^{y_{\text{tip}}} q_z \, dy'$$

$$M(y) = \int_y^{y_{\text{tip}}} V \, dy'$$

$$T(y) = \int_y^{y_{\text{tip}}} t(y') \, dy'$$

with distributed torsion $t = q_z \, (x_{cp} - u_{cg}) + m$ under the doubly-symmetric assumption $x_{sc} \equiv x_{cg}$. To determine stress components per station, we use bending:

$$\sigma_b = |M| \, z_{\max} / I_{uu}$$

with $z_{\max} = \sqrt{3}\sqrt{I_{uu}/A}$ (rectangular-section equivalent fibre distance, used as a placeholder rather than a surrogate output), transverse shear $\tau_s = |V| \, Q_{\max}/(I_{uu} \, t_{\text{skin}})$, and Bredt torsion $\tau_t = |T|/(2A_{\text{enc}} \, t_{\text{skin}})$ with $A_{\text{enc}} = 20 \, A$ pinned as a coarse thin-walled wingbox placeholder. We use composite von Mises stress for the Carbon-Fibre-Reinforced Polymer (CFRP) material: $\sigma_{vm} = \sqrt{\sigma_b^2 + 3(\tau_s + \tau_t)^2}$, summing the two shear contributions at a common point as a conservative envelope. Tip deflection is evaluated by double forward cumulative trapezoid of $M/(E \, I_{uu})$ from the cantilever root, returning the full $w(y)$ profile. The benchmark reads $w_{\text{tip}} = w(y_{\text{tip}})$ for the tip-deflection constraint. In terms of materials, we assume quasi-isotropic CFRP laminate with $E = 55$ GPa, $\rho = 1600$ kg/m^3, allowable strain $\varepsilon_{\text{allow}} = 4 \times 10^{-3}$. We are aware that these are gross simplifications, but we consider the above choices adequate for the application demo at hand.

Structural Mass. We use trapezoid: $m_{\text{struct}} = \rho_{\text{mat}} \int_0^{b/2} A_{\text{phys}}(y) \, dy$. For the range computation, we use battery mass: $m_{\text{bat}} = \rho_{\text{bat}} \, V_{\text{bat}}$ with $V_{\text{bat}} = (w \, d \, h) \, L^3$ from the 6-D battery design vector $\mathbf{x}_{\text{bat}} = (c_x, c_y, c_z, w, d, h)$ (the first three slots are placement coordinates, the last three are the box dimensions in unit-scale ratios). We use $\rho_{\text{bat}} = 2400$ kg/m^3 for an advanced silicon-anode/NMC pack. For the sake of simplicity and because the battery tends to be small, we do not evaluate overlap constraints between the battery and internal structures. Overall mass is then evaluated as:

$$m_{\text{tot}} = m_{\text{struct}} + m_{\text{bat}} + m_{\text{payload}}$$

with payload pinned at 3 kg.

K.1.8 CONSTRAINTS.

Overall, there are three constraints evaluated with the models described above. First, we use a lift balance equality constraint:

$$(W - L_{\text{lift}})/W = 0$$

with $W = m_{\text{tot}}\, g$ and $L_{\text{lift}} = C_L\, q_\infty\, S_{\text{ref}}$.

This single constraint is what implicitly bounds battery mass: a heavier battery raises W and forces the geometry to produce more lift, producing more drag, reducing the benefit in range. The cruise angle of attack α_{cr} is independently restricted to $\pm 3°$ as a hard design-vector box bound (not an inequality constraint).

Next, we evaluate a maximum material strain constraint, which is evaluated as per-station residuals $c_i = \sigma_{vm}(y_i)/(E\,\varepsilon_{\text{allow}}) - 1$, and aggregated as $\frac{1}{N}\sum_i \text{ReLU}(c_i)$. The aggregate is exactly zero if every station satisfies its strain bound, and the $K{=}1$ formulation gives a smaller Jacobian than one constraint per station, which eases compute requirements drastically for projection-based methods as described in § 5.

Lastly, we constrain tip deflection: $|w_{\text{tip}}|/(0.10 \cdot b/2) - 1 \leq 0$.

K.1.9 OBJECTIVE

For the objective, we use pure Breguet range:

$$R = (\eta_{\text{prop}}/g)\, E^*_{\text{bat}}\, (L/D)\, (m_{\text{bat}}/m_{\text{tot}})$$

with $\eta_{\text{prop}} = 0.8$, specific energy $E^*_{\text{bat}} = 300$ Wh/kg, and $L/D = C_L/C_D$ from $\Phi^\theta_{\text{aero}}$. The reported objective is $-R/R_{\text{ref}}$ with $R_{\text{ref}} = 10^7$ m, sized so ∇obj stays $\mathcal{O}(1)$. R_{ref} is a normalization constant for gradient scaling, not a physical target. Achieved R during optimization is roughly an order of magnitude smaller.

K.2 URBAN PLANNING

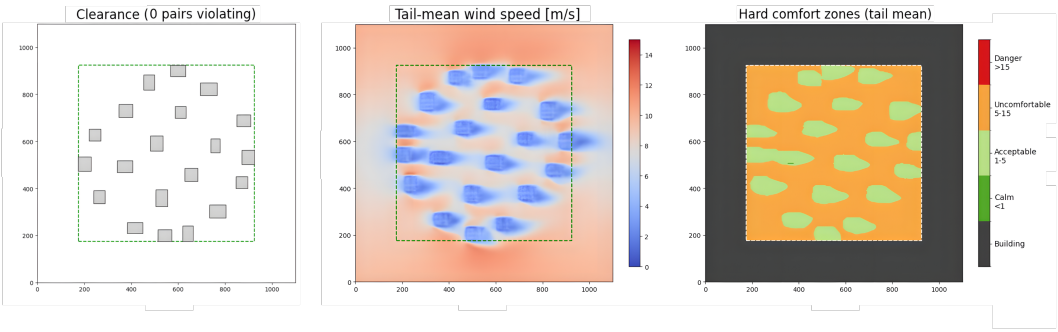


Figure 9: E2 urban planning benchmark, three views of a candidate design. **Left:** top-down site plan with the 750×750 m buildable plot (green dashed) and the 10 building footprints. Clearance is evaluated pairwise and reported as the number of pairs with overlapping $d_{\min}$-inflated AABBs. **Middle:** tail-mean wind speed predicted by the WinDiNet surrogate over the full 1100×1100 m raster domain. **Right:** hard pedestrian-comfort zones derived from the tail-mean speed (calm <1 m/s, acceptable 1–5, uncomfortable 5–15, danger >15). The danger constraint forbids any cell entering the red band.

Setup. Fig. 11 shows the arrangement of the evaluator. Place $B = 10$ rectangular buildings on a 750×750 m site to maximise total built volume under spacing, height, coverage, and pedestrian wind comfort constraints. The design variable y is 50D, with each building contributing $(c_x, c_y, w_{\text{raw}}, d_{\text{raw}}, h_{\text{raw}})$ and a softplus decoding on w, d, h to keep dimensions positive. The objective is $-\sum_i w_i\, d_i\, h_i/V_{\text{ref}}$ with $V_{\text{ref}} = 10^6$ m^3, which keeps the loss in $\mathcal{O}(1)$.

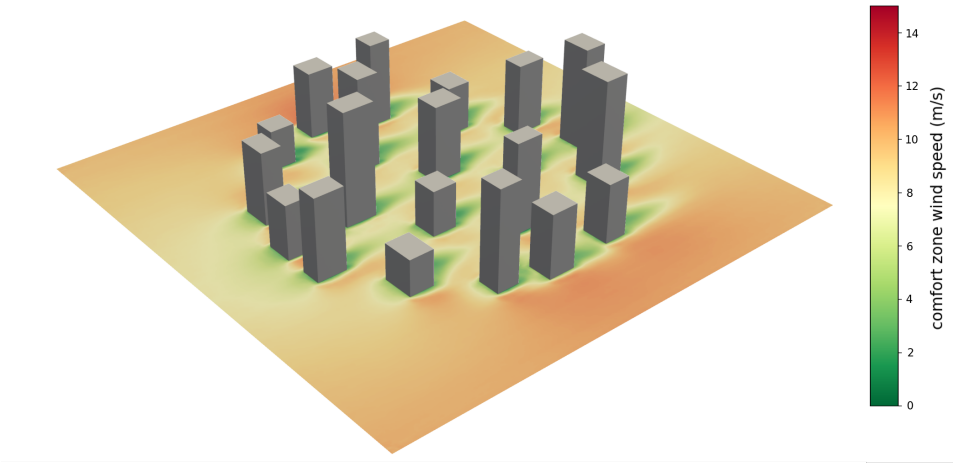


Figure 10: E2 candidate design rendered in 3D with PyVista (Sullivan & Kaszynski, 2019), with the pedestrian-comfort field draped at ground level. Building heights are decoded from the design vector. The colour map shows tail-mean wind speed over the comfort range 0–15 m/s, with green/yellow indicating calm and acceptable zones and red indicating the danger band that the surrogate constraint forbids.

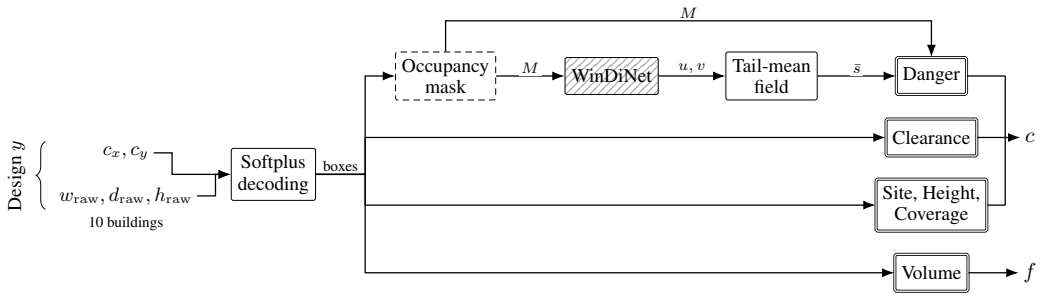


Figure 11: E2 evaluator Φ, from the design vector y to the objective f and constraints c. Dashed: geometry processing. Plain: analytical. Grey: neural surrogate. Hatched: generative model. Double border: constraint or objective.

Wind Surrogate. The pedestrian-comfort constraint is evaluated through WinDiNet (Perini et al., 2026), an LTX-Video 2B (HaCohen et al., 2024) diffusion backbone fine-tuned to roll out 2D pedestrian-level velocity fields over the full 1100×1100 m raster domain, conditioned on a building-footprint occupancy mask and a fixed inlet velocity. The surrogate is differentiable end-to-end, so gradients flow from each constraint back to the design vector as has been shown by Perini et al. (2026). We run 57 diffusion frames at 2 inference steps and score comfort on the tail-mean field over the last 10 frames, $\bar{s}(p) = \|(\bar{u}(p), \bar{v}(p))\|_2$.

Constraints. All five constraints are inequalities with per-constraint tolerance 10^{-4}.

- **Site** (footprint inside $[175, 925]$ m in y- and x-direction): softplus-smoothed squared-mean over the $4N$ box-edge primitives,

$$c_{\text{site}} = \frac{1}{4N} \sum_{i,e} \text{softplus}_\beta(p_{i,e})^2,$$

  with $p_{i,e} \in \{L_0 - x_i^-,\ x_i^+ - L_1,\ L_0 - y_i^-,\ y_i^+ - L_1\}$.
- **Height** (cap at $h_{\max} = 250$ m): $c_{\text{height}} = \frac{1}{B} \sum_i \text{relu}(h_i - h_{\max})$.

- **Coverage** (site-clipped footprint fraction ≤ 0.5): $c_{\text{cov}} = \text{relu}\big(\frac{1}{A_{\text{site}}} \sum_i A_i^{\text{clip}} - 0.5\big)$.
- **Danger** (no fluid cell exceeds 15 m/s in the tail mean): a fluid-mask–weighted mean of $\text{relu}(\bar{s} - 15)$ over inner-plot fluid cells, multiplied by $w_d = 100$ to bring the typical excess into the $\mathcal{O}(10^{-2})$ band. In raw m/s the cell-averaged excess sits below the 10^{-4} tolerance even when the integrated violation is non-trivial.
- **Clearance** (no pair within $d_{\min} = 10$ m): kept as $\binom{B}{2} = 45$ *per-pair* constraints,

$$c_{ij} = \text{relu}\big(\tfrac{w_i+w_j}{2} + \tfrac{d_{\min}}{2} - |c_x^{(i)} - c_x^{(j)}|\big) \cdot \text{relu}\big(\tfrac{d_i+d_j}{2} + \tfrac{d_{\min}}{2} - |c_y^{(i)} - c_y^{(j)}|\big),$$

  i.e., the m^2 overlap of the $d_{\min}$-inflated AABBs of pair (i, j).

We now provide some more details on why the last of these constraints, **Clearance**, was evaluated pairwise instead of aggregated into a mean or max score like the other constraints in this example.

Details on Pairwise Clearance Constraints. Aggregating the 45 pairwise overlaps into a single mean-pooled scalar weakened training across methods, which we believe is due to *gradient cancellation*: a building wedged between violating neighbours on opposite sides receives near-opposite push vectors that partially cancel under mean aggregation, so the gradient with respect to its center (c_x, c_y) is small even though the design is grossly infeasible. The per-pair formulation gives each pair its own multiplier and its own Jacobian row, restoring directional information and per-pair pressure. However, this comes at a cost, especially for projection-based methods: With B buildings, the number of **Clearance** constraints is $\frac{1}{2}B(B-1)$. Projection-based methods do need to build the constraint Jacobian J on each projection step, the cost of which scales with K (the overall number of constraints). However, in this case, this additional cost is attenuated by the fact that most constraints are such that only the graph relating to the **Clearance** constraint needs to be backpropagated back through, i.e., the expensive diffusion model for wind prediction is only backpropagated through once (for the **Danger** constraint), the cost of which dominates.

K.3 Power Grid Benchmark

`E3` is the AC Optimal Power Flow (ACOPF) benchmark, instantiated on the IEEE 57-bus system. Apart from the differences we list here, it is analogous to the application example in DC3 (Donti et al., 2021). ACOPF determines generator setpoints that meet bus-level demand at minimum generation cost while respecting the nonlinear physics of the AC power grid.

The conditioning vector $x = (p_d, q_d) \in \mathbb{R}^{2L}$ specifies real and reactive power demand at each load bus, and the design vector $y = (p_g, q_g, v_m, v_a) \in \mathbb{R}^{2G+2B}$ collects generator setpoints together with bus voltages in polar form (magnitude v_m, angle v_a). For IEEE-57 ($B = 57$ buses, $G = 7$ generators, $E = 80$ branches), this gives $\dim(y) = 128$. The objective is the standard quadratic generation cost,

$$f(y) = \sum_{i=1}^{G} \left(c_{0,i} + c_{1,i}\, p_{g,i} + c_{2,i}\, p_{g,i}^2\right), \tag{10}$$

normalised by $\bar{g}^2$, where $\bar{g}$ is the mean generator base power. This applies DC3's normalisation formula (the absolute scale differs slightly because pandapower's per-generator MBASE for the slack bus deviates from PYPOWER's). The equality constraints enforce real and reactive power balance at every bus ($2B = 114$ residuals, nonlinear in (v_m, v_a)). The inequality constraints box-bound generator outputs (p_g, q_g), bus voltage magnitudes, branch apparent power flows ($|S_{ij}|$ at both ends of each branch), and per-branch voltage angle differences. In total, we count 114 equalities and 462 inequalities, i.e., $K = 576$ constraints overall.

Differences from DC3 Implementation. Our variant of this benchmark is more difficult than the one in Donti et al. (2021), with a larger number of constraints. DC3's released code only enforces generator and voltage box bounds, $4G + 2B = 142$ inequalities. We additionally retain MATPOWER's (Zimmerman et al., 2011) thermal line limits and angle-difference bounds, $4E = 320$ extra inequalities, bringing the total to 462. On IEEE-57 these added slots are mostly inactive: the 160 angle-difference residuals are vacuous because the case data marks every branch as angle-unconstrained ($\pm 360°$) and our output box bounds each $v_a \in [-\pi, \pi]$, so $|v_{a,i} - v_{a,j}| \leq 2\pi$

holds by construction. The 160 thermal residuals are well-defined but slack ($S_{ij}^{\max} = 99$ pu after `baseMVA` normalisation), so they almost never bind on sampled OPF solutions. They do become active on extreme off-manifold predictions and therefore still contribute to the feasibility metric. For grid data we use `pandapower` (Thurner et al., 2018) and for constraint residuals we use `ML4OPF` (AI4OPT, 2024), whereas DC3 calls PYPOWER (Lincoln, 2011) directly. The underlying MATPOWER (Zimmerman et al., 2011) cases and admittance physics are aligned to within $\sim 10^{-5}$ pu, but pandapower's per-generator MBASE and slack-bus bound conventions differ slightly from PYPOWER's, requiring small loader-side adjustments. Our condition pool is regenerated under our loader: per-sample correlated uniform load perturbations around the nominal demand, retaining only samples for which an offline ACOPF solve with thermal limits relaxed converges. Every method evaluated here, including the DC3 baseline, draws its conditions from this same pool, so the head-to-head comparisons in this paper are internally fair in terms of conditioning.

Cost and Hardware. With $K = 576$ (114 equalities and 462 inequalities), this is the largest constraint count among our engineering benchmarks. Per training step, PAL's cost is dominated by Jacobian assembly, one backward pass through Φ per constraint row, while DC3 assembles the Jacobian over the 114 equalities only but takes several inner correction steps per training step, which results in higher wall-clock times overall. ALM's per-step cost carries no dependence on K (one forward and one backward through Φ), which is why it trains in about a minute. PAL finishes roughly 2.5 times faster than DC3. This benchmark was run on a 288-core ARM CPU node. Cross-method ratios are reliable, but absolute wall-clock numbers are hardware-specific.

L Failure Modes of Baselines in Engineering Benchmarks

This appendix collects some of our hypotheses for the failure modes observed on `E1`–`E3` for the baselines. We do not study each failure in depth.

DC3 on `E1` and `E2`. DC3 is structurally inapplicable here. Its completion step solves the equality constraints for a dependent subset of the outputs and requires a known dependence on that subset. The single equality constraint of `E1` and `E2` is a neural surrogate of the full design vector, so no such subset exists, and DC3 would reduce to a non-adaptive penalty loss followed by gradient-correction steps. We therefore do not run it on `E1` and `E2`.

IPOPT on `E2`. No IPOPT restart converges on `E2`. The best iterate of every restart is feasible, but only 31% of final iterates are, and all restarts terminate with a local-infeasibility status. The aggregated constraints of `E2` apply ReLU transforms, so the Jacobian rows of inactive constraints vanish identically and the constraint qualification fails at most iterates (95.6% of Jacobian rows are zero at the final iterates, and the remaining rows span roughly 35 orders of magnitude). We speculate that this is the pathology Yang et al. (2022) report for IPOPT on embedded ReLU networks, where dual infeasibility is never driven to zero and the solver terminates at spurious points. On `E1`, IPOPT reaches feasible points in every restart but also never exits with a converged status.

SnareNet on `E3`. On `E3`, the number of constraints $K = 576$ is much larger than $\dim(y) = 128$. We believe that SnareNet was conceived around $K \leq \dim(y)$, since its convergence proofs require a constraint Jacobian of full row rank (Chu et al., 2026). In Appendix G of the original SnareNet paper, Chu et al. (2026) show experiments where SnareNet is trained under $K > \dim(y)$. They show that, on synthetic non-convex programs with $K = 500$ and $\dim(y) = 100$, SnareNet exhausts its projector budget of 100 steps without converging. However, we have no intuition for why this occurs, as Chu et al. (2026) do not elaborate on the suspected reasons for this failure beyond noting that SnareNet suits problems with $K \ll \dim(y)$. On `E3`, each repair step reduces the relaxed residual by roughly 2%, so we cap the repair budget at 5 steps, and all ten seeds diverge to non-finite losses within the first 300 epochs.

ENFORCE on `E3`. ENFORCE provides an optional gate during training. This gate rejects any projection step in training if it does not lower the soft loss. We find that, without the gate, training diverges catastrophically on `E3`, with all ten seeds producing a non-finite loss in the first epoch. The gate is therefore needed to stabilize training. With the gate enabled, however, we observe that

all projection steps on E3 are rejected. Effectively, ENFORCE then trains without any projection awareness. At test time, the projection is applied regardless, and we then observe divergence, resulting in numerical errors on 2 to 13 of the 64 test queries per seed.

FSNet on E3. FSNet seeks feasibility by minimizing the summed constraint violation with L-BFGS. We speculate that this is much weaker than projection on E3 because of the extremely high number of constraints. Amalgamating them into a single score discards the per-constraint information that a full Jacobian retains. Additionally, the single score assumes somewhat comparable scales across constraints, but ACOPF constraints (voltages, thermal limits, ...) have mixed units and potentially very different scales. The FSNet paper itself does not use L-BFGS on its ACOPF benchmark but switches to LM for the feasibility-seeking step (Nguyen & Donti, 2025, Appendix B), which requires the constraint Jacobian in every step. This variant is not part of the public code, so our port runs the generic L-BFGS solver on E3. We read the authors' switch as confirmation that first-order feasibility seeking is not viable on E3.

ENFORCE on E1 and E2. We run variants with and without penalty-method warm-up. On E1 and E2, and we used the IFT variant for better memory efficiency, but find it does not reduce peak memory usage. 39 of 40 seeds run OOM while assembling this Jacobian. On E1, a single ENFORCE seed finishes training, but runs OOM in evaluation due to a longer repair unroll.

M Hyperparameter Tuning

All baselines and PAL were tuned with Bayesian optimization on the synthetic suite S1–S6. Generally, the budget per method was set to 8 Sobol and 24 Bayesian-optimization trials, i.e., 32 trials per method, with 3 seeds per trial. Two exceptions are ALM+Bolt-On (8+12 due to the very low-dimensional search space) and FSNet (stopped early due to plateau and very long run times). No sweep improved on its best trial after trial 24. The top three trials per method were then re-run with 8 seeds, and the winner was selected lexicographically: highest feasibility, then lowest objective, then lowest constraint violation. The ten reported seeds in the results are disjoint from the tuning seeds and were never used in selection. A run counts as diverged if it crashes catastrophically or its post-repair objective exceeds 10^2. Table 8 shows the tuned hyperparameters per method.

Table 8: Selected hyperparameter values per method from the Bayesian optimization sweep. †: For ALM+Bolt-On, the optimized hyperparameters of frozen ALM models were reused, i.e., only hyperparameters of the projection were tuned for ALM+Bolt-On itself.

Method	BO winner configuration
ALM	lr $8.3{\cdot}10^{-5}$; γ $8.8{\cdot}10^{-4}$; α 0.995; eps 10^{-4}
ALM+Bolt-On	proj_delta 10^{-5}; proj_max_iters 20 †
ENFORCE (v4)	lr $2.5{\cdot}10^{-4}$; eps_chol $2.3{\cdot}10^{-5}$; max_it 100; training_tolerance 0.01; weight_loss_displacement 5.0; weighting_option 5; ada_np_auto_activation true (fixed: epoch_start_hard_constrained 0, eps 10^{-8}, inference_tolerance 10^{-6})
DC3	lr $1.75{\cdot}10^{-5}$; soft_weight 10; corr_train_steps 10; corr_lr $1.8{\cdot}10^{-6}$
FSNet	lr $2.0{\cdot}10^{-4}$; dist_weight 5.0; weight_decay $3.0{\cdot}10^{-5}$
SnareNet	lr $2.5{\cdot}10^{-4}$; λ $2.6{\cdot}10^{-3}$; soft_weight 1.0; soft_epochs 1000; decay_epochs 1000
PAL [Ours]	lr $7.6{\cdot}10^{-5}$; rate $1.1{\cdot}10^{-3}$; max_decades 5.0

N Reproducibility

Here, we state the hardware used in our experiments and provide rough wall-clock times, and then show the hyperparameters used for PAL and ALM in the engineering benchmarks. Hyperparameters on the synthetic benchmarks come from the equal-budget Bayesian optimization sweep in

Appendix M. On E1–E4 we generally use paper defaults. FSNet, SnareNet, and ENFORCE hyperparameters are not shown for the engineering benchmarks E1 and E2. These were not tuned since these methods consistently fail immediately due to memory intractability anyway.

On E3, all learned methods follow the published DC3 schedule for ACOPF, i.e., a 200×2 backbone, 1000 training samples, 2000 epochs, batch size 200, and learning rate 10^{-3}. DC3 runs at its published hyperparameters, PAL at defaults with $\lambda_{\min} = 10^{-4}$. FSNet and ENFORCE otherwise run at their published defaults. SnareNet runs at its published defaults except for a repair budget of 5 steps, since its repair layer reduces the residual by roughly 2% per step and 100 steps would exceed 200 hours per seed.

N.1 Hardware and Wall-Clock

Runs were generally executed on GH200 Grace-Hopper nodes. The engineering benchmarks E1 and E2 used 4× GH200 per seed via DDP. Synthetic benchmarks (S1–S6), E3, E4, and ablations were run CPU-only. Wall-clock times for E3 and E4 are reported separately in Appendices K.3 and J.

Tables 9 and 10 report median end-to-end wall-clock per run over 10 seeds (n=5 where noted in the corresponding results tables). These figures cover configuration, training, final evaluation, and artifact saving. Training dominates in every cell shown.

N.2 Hyperparameters

For the neural backbone N_θ, all methods use Adam with a simple MLP 512×4 prediction head. ALM+Bolt-On shares ALM's training runs for the engineering benchmarks E1 and E2. At inference, it adds a single Levenberg–Marquardt projector pass that uses PAL's projector defaults (N=10, δ=10^{-3}). The benchmark sets the residual gap target τ_c used by PAL's log-gap rule (τ_c=10^{-3} on E1 and τ_c=10^{-4} on E2). We treat τ_c as a property of the benchmark, not a method-specific hyperparameter. For decay, ✓ marks the bidirectional log-gap update (the default, where multipliers $\bar{w}_{c,k}$ and $\bar{w}_d$ can decay), and ✗ marks monotone-only growth, which we enable only on E1 since we observe moderate oscillations between $\bar{w}_c$ and $c(\tilde{y}^{(1)})$ which we want to avoid. We are unsure what causes these.

FSNet runs the authors' implementation with its default truncated unrolled differentiation, unrolling up to 50 repair steps and backpropagating through the first 30, which lies inside the recommended range (10 to 50) reported in the FSNet ablations (Nguyen & Donti, 2025). Like this default, all remaining FSNet settings are left at the authors' values, apart from the tuned hyperparameters listed in Appendix M.

O Compute Scaling

In this appendix, we provide some thoughts on how PAL performs on problems with a large number of constraints, and how complexity scales with problem dimensionality, the number of constraints, and constraint cost heterogeneity. Table 12 shows the full results table of the synthetic benchmarking results, including full N_b^ϕ and N_f^ϕ, which Table 1 showed only excerpts of.

Cost Model. Per training step, PAL costs $2C_f + (K+1)C_b$ evaluator passes, where C_f and C_b are the costs of one forward and one backward pass (VJP) through the evaluator Φ, respectively. This is because two forward passes are needed, one before the projection at $\hat{y}$ and one after the projection at $\tilde{y}^{(1)}$. K VJP backward passes are needed to build the Jacobian J, and one backward pass is needed to backpropagate the training loss (weighted constraints and the objective).

For the special case of $K \gg \dim(y)$, J can be assembled from forward-mode JVPs instead of reverse-mode VJPs, meaning the assembly cost is $\min(K, \dim(y))$ passes. Additionally, PAL requires $O(K^2 \dim(y) + K^3)$ for the damped normal system of the LM projector. In the regime we target, the evaluator passes dominate and the latter is negligible. The recorded number of constraint evaluator calls in Table 1 fits this model exactly (e.g., 91 backward calls at $K = 90$ on S6).

Table 9: Wall-clock times on S1–S6. The total time reported includes initialization, training, evaluation, and artifact saving. The overall prediction time is for 64 samples (queries). We report the median over 10 seeds (counting non-diverged seeds only). "—" indicates fully diverged or structurally inapplicable cells. †For ALM+Bolt-On, training reuses ALM checkpoints and the reported total time is inference-time projection only.

	Method	Train (s)	Total (s)	Predict (s)	/query (ms)
S1	**PAL**	106	247	0.0119	0.185
	ALM	58	131	0.00734	0.115
	ALM+Bolt-On	58†	10	0.0507	0.792
	ENFORCE v4	57	63	0.0152	0.237
	FSNet	1720	1909	0.176	2.75
	SnareNet	149	310	0.0506	0.791
	DC3	—	—	—	—
S2	**PAL**	118	261	0.018	0.282
	ALM	65	136	0.00917	0.143
	ALM+Bolt-On	65†	49	0.0627	0.98
	ENFORCE v4	104	110	0.0422	0.659
	FSNet	1881	2074	0.0395	0.617
	SnareNet	152	292	0.175	2.73
	DC3	261	417	0.0166	0.259
S3	**PAL**	126	276	0.0159	0.249
	ALM	71	126	0.00775	0.121
	ALM+Bolt-On	71†	50	0.408	6.37
	ENFORCE v4	72	80	0.0448	0.700
	FSNet	1593	1791	0.0174	0.272
	SnareNet	216	277	0.156	2.44
	DC3	253	410	0.0213	0.333
S4	**PAL**	179	339	0.0485	0.758
	ALM	91	196	0.00776	0.121
	ALM+Bolt-On	91†	52	0.367	5.74
	ENFORCE v4	183	189	0.950	14.8
	FSNet	8045	8268	0.0688	1.08
	SnareNet	460	483	0.788	12.3
	DC3	—	—	—	—
S5	**PAL**	135	303	0.023	0.36
	ALM	116	260	0.00744	0.116
	ALM+Bolt-On	116†	34	0.094	1.47
	ENFORCE v4	—	—	—	—
	FSNet	15772	15968	0.238	3.73
	SnareNet	142	167	0.14	2.19
	DC3	—	—	—	—
S6	**PAL**	367	516	0.332	5.19
	ALM	77	227	0.0137	0.214
	ALM+Bolt-On	77†	41	0.736	11.5
	ENFORCE v4	1243	1248	0.432	6.75
	FSNet	1794	2042	0.271	4.23
	SnareNet	1493	1542	5.79	90.5
	DC3	—	—	—	—

Comparison with Full-Unroll Projection Methods. Every projection-based method (DC3, ENFORCE, SnareNet) assembles the same K-row Jacobian, but unrolled methods do so at each of N repair iterates. This results in an overall cost of $N(C_f + KC_b)$ versus PAL's single assembly. As disclosed in item (3) of the Limitations section, this cost is shared by all projection-based methods, and PAL is the cheapest member of that class.

Table 10: End-to-end wall-clock time on engineering benchmarks. On E2, PAL suffers from sequential Jacobian assembly (loop over K).

Benchmark	Method	Median wall-clock
E1 BWB	**PAL**	∼8 min
	ALM	∼70 s
E2 Urban Planning	**PAL**	∼106 min
	ALM+Bolt-On	∼65 min
	ALM	∼32 s

Table 11: Hyperparameters for PAL and ALM on E1 and E2. Bold marks deviations from defaults. ALM defaults follow Basir & Senocak (2023). For both benchmarks, we performed minor hand-tuning of γ_{ALM} and α but found defaults to work relatively well.

	Default	E1 (Unmanned Aircraft)	E2 (Urban Planning)
Training epochs	—	200	500
Batch size	—	32	16
Learning rate (N_θ)	—	10^{-4}	10^{-3}
PAL			
γ	10^{-2}	$\mathbf{10^{-1}}$	$\mathbf{10^{-1}}$
$D_{\max}$	1.0	**5.0**	**5.0**
τ_d	10^{-4}	10^{-4}	10^{-4}
δ	10^{-3}	10^{-3}	10^{-3}
$\lambda_{\min}$	10^{-6}	10^{-6}	10^{-6}
Decay	✓	✗	✓
ALM			
γ_{ALM}	10^{-2}	$\mathbf{10^{-3}}$	10^{-2}
α	0.99	0.99	0.99
μ_0	1.0	1.0	1.0
λ_0	1.0	1.0	1.0

Comparison with FSNet. FSNet sidesteps Jacobian assembly and requires only $N(C_f + C_b)$, but unrolls many more repair steps (large N). The break-even point between PAL and FSNet is roughly at $K \approx 2N$. FSNet must still store the unrolled tape, which can be memory-intractable (see Table 2).

Heterogeneous Constraint Complexity. In some applications, one or more constraints might be extremely expensive to compute and dominate compute cost. For example, E2's wind-speed danger constraint shown in Appendix K.2 is evaluated using diffusion model unrolls, while other constraints in this problem are trivial to compute (e.g., box-to-box overlaps, analytical closed-form constraints, etc.). In this situation, the Jacobian assembly count K virtually drops out of PAL's cost while FSNet still pays the full unroll.

Other than cost, we do not see any specific reasons PAL might struggle with large-K. Benchmark E3 in Table 2 illustrates that PAL can be applied without any modifications to applications with a very high number of constraints ($K = 576$) and trains faster than DC3 thanks to single-step training, despite building the full Jacobian (which DC3 does not, as it only projects equalities).

Table 12: Full version of Table 1: synthetic stress-test experiments for five baselines and PAL, including evaluator passes per training step on all benchmarks. Means and standard deviations across 10 seeds. "—" indicates failure on 2 or more seeds.

	Bench.	ALM+ Bolt-On	ENFORCE	DC3	FSNet	SnareNet	PAL [Ours]
Feas. (%)	S1	100 ± 0	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	S2	100 ± 0	100 ± 0	97 ± 7	100 ± 0	100 ± 0	100 ± 0
	S3	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0	100 ± 0
	S4	100 ± 1	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	S5	100 ± 0	—	—	51 ± 31	100 ± 0	100 ± 0
	S6	100 ± 0	100 ± 0	—	100 ± 0	100 ± 0	100 ± 0
	Mean	**100**	—	—	92	**100**	**100**
Obj.	S1	0.07 ± 0.12	0.00 ± 0.00	—	0.00 ± 0.00	0.78 ± 0.10	0.03 ± 0.04
	S2	0.27 ± 0.17	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.16 ± 0.00	0.00 ± 0.00
	S3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.00 ± 0.00
	S4	19.47 ± 5.25	0.01 ± 0.01	—	0.00 ± 0.00	0.09 ± 0.01	0.00 ± 0.00
	S5	0.00 ± 0.00	—	—	4.41 ± 2.16	0.00 ± 0.00	0.00 ± 0.00
	S6	0.14 ± 0.03	0.00 ± 0.00	—	0.00 ± 0.00	0.60 ± 0.06	0.03 ± 0.01
	Mean	3.32	—	—	0.73	0.27	**0.01**
N_Φ^{f}	S1	1.0 ± 0.0	6.2 ± 0.1	—	72 ± 77	35.0 ± 5.0	2.0 ± 0.0
	S2	1.0 ± 0.0	6.0 ± 0.0	34.5 ± 0.0	18.5 ± 0.1	20.5 ± 1.0	2.0 ± 0.0
	S3	1.0 ± 0.0	5.2 ± 0.1	34.5 ± 0.0	10.1 ± 0.1	44.1 ± 5.9	2.0 ± 0.0
	S4	1.0 ± 0.0	6.4 ± 0.0	—	77.7 ± 0.6	61.9 ± 2.8	2.0 ± 0.0
	S5	1.0 ± 0.0	—	—	595 ± 50	50.3 ± 3.6	2.0 ± 0.0
	S6	1.0 ± 0.0	7.1 ± 0.1	—	15.5 ± 0.0	81.1 ± 2.9	2.0 ± 0.0
	Mean	**1.0**	—	—	131	48.8	2.0
N_Φ^{b}	S1	1.0 ± 0.0	4.1 ± 0.1	—	11.0 ± 5.6	33.6 ± 5.0	2.0 ± 0.0
	S2	1.0 ± 0.0	14.1 ± 0.0	22.0 ± 0.0	7.8 ± 0.1	22.7 ± 1.3	12.0 ± 0.0
	S3	1.0 ± 0.0	7.4 ± 0.2	22.0 ± 0.0	5.6 ± 0.1	47.7 ± 6.7	7.0 ± 0.0
	S4	1.0 ± 0.0	32.5 ± 0.2	—	39.1 ± 0.2	96.9 ± 5.3	27.0 ± 0.0
	S5	1.0 ± 0.0	—	—	87.8 ± 1.0	53.4 ± 4.1	6.0 ± 0.0
	S6	1.0 ± 0.0	128 ± 2	—	7.2 ± 0.0	253 ± 12	91.0 ± 0.0
	Mean	**1.0**	—	—	26.4	84.6	24.2