FLEXCONE: AN ALWAYS-FEASIBLE FIRST-ORDER SOLVER FOR CONIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Conic subproblems produced by linearization or convexification, as in sequential convex programming (SCP) and related methods, are frequently infeasible, even when the original nonlinear constraints are satisfiable. Existing solvers certify such infeasibility but return nothing to act on, so controllers fall back on stale solutions that violate constraints. We propose FlexCone, an always-feasible first-order conic solver that handles feasible and infeasible problems within a single framework. FlexCone applies ADMM to an exact relaxation of the conic program that provably recovers the original solution when the problem is feasible and converges to a minimum-violation point when it is not. Because every instance runs the same operations, FlexCone batches efficiently on a GPU and can be trained using deep unfolding. On standard convex benchmarks, learned FlexCone matches or improves upon learned projection-based solvers. In closed-loop model predictive control, it is competitive on feasible subproblems and recovers from infeasible ones, eliminating the constraint violations and collisions of the standard fallback. On neural network robustness certification, where infeasibility of an LP relaxation certifies robustness, our batched solver returns certificates orders of magnitude faster than baseline solvers on the same formulations.

1 INTRODUCTION

Many nonlinear decision-making problems in robotics, power systems, and machine learning are solved by repeatedly convexifying them and passing the resulting conic subproblem to a convex solver, as in sequential quadratic programming (SQP), sequential convex programming (SCP), and successive convexification (SCvx) (Nocedal & Wright, 2006; Mao et al., 2016; Malyuta et al., 2022). Linearizing the constraints and adding a trust region can make a subproblem infeasible, even when the original nonlinear constraints are satisfiable. We emphasize that this is not an edge case: in the closed-loop experiments of Section 3.2, a standard first-order solver certifies 69% of the subproblems infeasible on one task, and on another the very first linearization is infeasible on most scenarios, although every nonlinear problem is solvable. When this happens, modern conic solvers, both interior-point (Goulart & Chen, 2024) and first-order (O'Donoghue et al., 2016; Banjac et al., 2019; Garstka et al., 2021), return a certificate of infeasibility: a dual ray proving that no feasible point exists. The certificate is mathematically rigorous but gives the outer algorithm nothing to act on, since it contains no primal iterate or direction that would reduce the violation. Practitioners therefore either reuse the previous stale solution, which in closed-loop control can violate state constraints or cause collisions, or rewrite the subproblem by hand with extra slack variables and penalty weights (Gill et al., 2005; Mao et al., 2016). Both workarounds lie outside the solver on a separate code path that is taken only for some instances. A separate code path is awkward to batch on a GPU and cannot be differentiated through, and both of those are necessary for the high-throughput, large-scale solvers that are becoming popular across many fields and applications.

In this work, we propose FlexCone, an always-feasible first-order solver for general conic optimization. FlexCone replaces each conic constraint with a penalty on the distance to its cone, weighted by a per-block parameter μ_i, and applies the alternating direction method of multipliers (ADMM) to the penalized problem. The penalty is *exact*: whenever μ_i exceeds the dual norm of the optimal multiplier of block i, the penalized and original problems have the same solutions (Theorem 2.3). This extends the classical ℓ_1 result for linear constraints (Pietrzykowski, 1969; Han & Mangasarian, 1979) to second-order and semidefinite cones, each with its own exactness condition and closed-form prox

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.

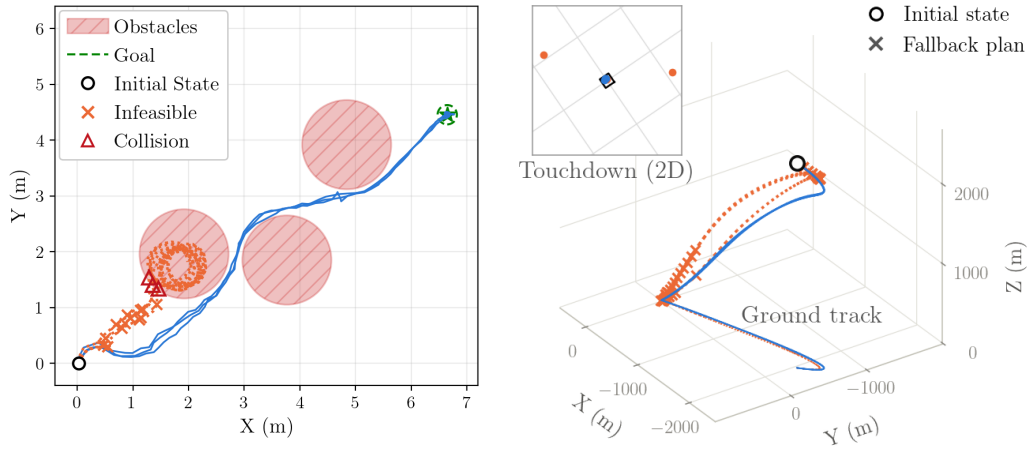


Figure 1: Closed-loop sequential convex programming scenarios. COSMO (orange, dotted) consistently falls back to its previous plan, resulting in suboptimal trajectories that collide with obstacles and fail the task. FlexCone (blue, solid) retains robust control over the vehicle via its unified infeasibilty handling. Three noise seeds are plotted per algorithm. Left: iterative covariance steering. Right: powered descent guidance.

(Table 1). The only change from COSMO (Garstka et al., 2021) is that the hard projection onto each cone becomes the prox of its distance penalty, which costs no more than the projection itself and reduces to it whenever the multiplier stays below the penalty, so, on feasible problems, FlexCone takes the same steps as COSMO. On an infeasible problem, the dual iterates remain bounded by μ instead of diverging along the certificate, and the method converges to a bounded KKT point of the penalized problem (Corollary 2.7). This point trades the objective against a weighted violation of the linearized constraints and, for large penalties, minimizes the violation (Proposition 2.5). This gives the outer algorithm a step it can act on, and the violated blocks identify which constraints carry the infeasibility. One set of optimality conditions and one termination test therefore cover both feasible and infeasible instances alike, and a final check on the multipliers certifies whether the returned point solves the original problem, without knowledge of the optimal dual multiplier.

Because each instance runs the same operations, FlexCone can be batched efficiently on a GPU and differentiated through via unrolling. We exploit this to learn hyperparameter and warm-start policies using deep unfolding (Oshin et al., 2026b). The policies are trained with a self-supervised loss on the penalized optimality conditions that requires no labeled solutions and remains well-defined on infeasible instances, where diverging multipliers of existing solvers would leave a supervised loss with no finite target. We evaluate FlexCone on a suite of problems and applications across optimization, control, and machine learning (Section 3). Our contributions are summarized as follows.

- **Exact conic relaxation.** A cone-distance penalty with per-block dual-norm exactness conditions, closed-form prox operators for polyhedral, second-order, and semidefinite cones at the cost of a projection, and convergence of the resulting ADMM to a bounded KKT point on feasible and infeasible problems alike (Section 2).
- **Learned FlexCone.** On six feasible benchmark classes, learned FlexCone matches or improves on learned COSMO in median iterations on every class, and is stable on the Lovász ϑ SDPs, where learned COSMO is slower than its own untrained baseline.
- **Closed-loop control.** On receding-horizon covariance steering and powered descent, COSMO's fallback to a stale plan causes collisions and failed landings, whereas FlexCone's minimum-violation step completes most episodes with fewer ADMM iterations. On 6-DoF powered descent SCP it solves every scenario at a fraction of the cost of the virtual-control formulation.
- **Robustness certification.** On LP relaxations whose infeasibility certifies classifier robustness, batched FlexCone returns no wrong verdicts, decides infeasible problems within tens of iterations, and is one to two orders of magnitude faster than serial solvers. Its returned point attains the exact sparse minimum violation over a handful of constraints, while COSMO's is uninformative.

2 FLEXCONE: ALWAYS-FEASIBLE CONIC OPTIMIZATION

2.1 CONIC OPTIMIZATION

We consider the standard-form conic program:

$$\underset{x}{\text{minimize}} \quad f(x) := \tfrac{1}{2}x^\top Px + q^\top x, \quad \text{subject to} \quad b - Ax \in K, \tag{1}$$

where $P \in \mathbb{S}^n_+$, $q \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and K is a non-empty, closed, convex cone. K itself can be a Cartesian product of M convex cones, $K = K_1 \times \cdots \times K_M$, such that $b_i - A_i x \in K_i$ for $i = 1, \ldots, M$, with $A_i \in \mathbb{R}^{m_i \times n}$ and $b_i \in \mathbb{R}^{m_i}$, where m_i is the size of block i. This allows Eq. (1) to model an arbitrary combination of conic constraints, such as polyhedral, second-order cone, and positive semidefinite constraints. We define the slack as $s := b - Ax$, with individual blocks $s_i = b_i - A_i x$.

The KKT conditions for Eq. (1) are given by:

$$Px^* + q + A^\top y^* = 0, \quad b - Ax^* \in K, \quad y^* \in K^*, \quad \langle y^*, b - Ax^* \rangle = 0, \tag{2}$$

where K^* is the dual cone of K and $\mathcal{L}(x, y) := f(x) + \langle y, Ax - b\rangle$ is the associated Lagrangian. A point (x^*, y^*) satisfying the KKT conditions Eq. (2) solves Eq. (1). We do not assume that Eq. (1) is feasible in general. When it is feasible, we will assume a constraint qualification holds (e.g., Slater's condition) such that a KKT point exists.

2.2 EXACT RELAXATION

We let each block i be equipped with a norm $\|\cdot\|$ on $\mathbb{R}^{m_i}$, with corresponding dual norm $\|\cdot\|_*$ and dual unit ball $B_i^* := \{u \mid \|u\|_* \le 1\}$. We define the distance to the cone K_i as $\operatorname{dist}_{K_i}(s_i) := \min_{p\in K_i} \|p - s_i\|$. For notational conciseness, we omit the block index on the norms and note that the norm may differ between blocks. The choice of norm depends on the type of cone being considered, as we will discuss in Section 2.3. The relaxed problem replaces each conic constraint by a penalty based on the distance to the cone:

$$\underset{x}{\text{minimize}} \quad \phi_\mu(x) := f(x) + \sum_{i=1}^{M} \mu_i \operatorname{dist}_{K_i}(b_i - A_i x), \tag{3}$$

where $\mu = (\mu_1, \ldots, \mu_M)$ with $\mu_i > 0$. For separable cones (zero cone, nonnegative orthant) we use the ℓ_1 norm, whose prox is an elementwise soft-threshold. For non-separable cones (SOC, PSD) we use the ℓ_2 or Frobenius norm, which preserves the geometry of the cone and yields a prox that costs no more than the equivalent projection. Let $K_i^\circ = -K_i^*$ denote the polar cone. We define the set of *truncated multipliers* as

$$D_\mu := D_{\mu_1} \times \cdots \times D_{\mu_M}, \quad D_{\mu_i} := \{y_i \in K_i^* \mid \|y_i\|_* \le \mu_i\}.$$

We say block i is *saturated* at y_i if $\|y_i\|_* = \mu_i$. All proofs for this section are given in Appendix A.

Lemma 2.1 (Dual representation)**.** *For each block i and for any $s_i \in \mathbb{R}^{m_i}$, $y_i \in K_i^*$, we have that*

$$\operatorname{dist}_{K_i}(s_i) = \max_{u \in B_i^* \cap K_i^\circ} \langle u, s_i\rangle \quad \textit{and} \quad -\langle y_i, s_i\rangle \le \|y_i\|_* \operatorname{dist}_{K_i}(s_i). \tag{4}$$

Consequently, $\phi_\mu(x) = \max_{y\in D_\mu} \mathcal{L}(x, y)$.

The relaxation therefore keeps the same Lagrangian of Eq. (1) but truncates its multipliers to the compact set D_μ. Every result below follows from this observation.

Proposition 2.2 (Penalized optimality conditions)**.** *A point $\hat{x}$ minimizes Eq. (3) if and only if there exists $\hat{y}$ such that*

$$P\hat{x} + q + A^\top \hat{y} = 0, \quad \hat{y} \in D_\mu, \quad \langle \hat{y}_i, A_i\hat{x} - b_i\rangle = \mu_i \operatorname{dist}_{K_i}(b_i - A_i\hat{x}), \ \ \forall i = 1, \ldots, M. \tag{5}$$

Moreover, every violated block is saturated, namely, if $\operatorname{dist}_{K_i}(b_i - A_i\hat{x}) > 0$, then $\|\hat{y}_i\|_* = \mu_i$.

When $b - A\hat{x} \in K$, the last condition in Eq. (5) reduces to complementary slackness, and Eq. (5) reduces to the KKT conditions Eq. (2). This single set of conditions thus captures both feasible and infeasible problems, and we build our training loss on it in Section 2.4. A minimizer exists when $P \succ 0$ or, more generally, when μ dominates the decrease of the objective along every direction in the nullspace of P (Proposition A.1); the relaxation removes infeasibility but not unboundedness. A trust region constraint, commonly used in SCP, guarantees this for μ large enough relative to q.

Theorem 2.3 (Exactness). *The following two results hold:*

(a) Let (x^*, y^*) be a KKT point of Eq. (1). If $\mu_i > \|y_i^*\|_*$ for every block i, then the solution sets of Eq. (1) and Eq. (3) coincide.

(b) If $(\hat{x}, \hat{y})$ satisfies Eq. (5) with no saturated block, then it is a KKT point of Eq. (1).

Part (a) extends the classical exactness condition of the ℓ_1 penalty, $\mu > \|y^*\|_\infty$ (Pietrzykowski, 1969; Han & Mangasarian, 1979), block-wise to general cones, replacing $\|\cdot\|_\infty$ with the dual norm of each block's penalty (Table 1). The inequality must be strict (see the remark after the proof in Appendix A). Part (b) serves as a useful certification step: if no block is saturated at the final iterate, it must solve Eq. (1), without knowledge of y^* and without any constraint qualification.

Corollary 2.4 (Infeasibility and saturation). *If Eq. (1) is infeasible, every pair $(\hat{x}, \hat{y})$ satisfying Eq. (5) has at least one saturated block.*

Together with Proposition 2.2, this identifies the constraints responsible for an infeasibility. Since $\hat{x}$ satisfies every block that is not violated, deleting the violated blocks restores feasibility, so they intersect every irreducible infeasible subsystem (IIS) (Chinneck, 2008), although they need not form an infeasible subsystem themselves.

Proposition 2.5 (Minimum-violation point). *Fix weights $w_i > 0$ and let $V(x) := \sum_i w_i \operatorname{dist}_{K_i}(b_i - A_i x)$ be the weighted violation, with minimum value $V_{\min} := \inf_x V(x)$. Setting $\mu = tw$ gives $\phi_{tw} = f + tV$, and for each $t \geq t_0$ let x_t be a minimizer of $f + tV$. As t increases, the violation $V(x_t)$ is nonincreasing and the objective $f(x_t)$ is nondecreasing. As $t \to \infty$, $V(x_t) \to V_{\min}$, and every limit point of $\{x_t\}$ solves $\min_x f(x)$ subject to $V(x) = V_{\min}$.*

This is a classical result for penalty methods and shows that for finite μ the minimizer trades off objective and weighted violation, and is a minimum-violation point in the limit $t \to \infty$.

2.3 Algorithm

To derive the FlexCone algorithm, we solve Eq. (3) using ADMM. We adopt the operating splitting from OSQP (Stellato et al., 2020), which results in a more numerically favorable algorithm, by introducing $z = b - Ax$ and copy variables $(\tilde{x}, \tilde{z}) = (x, z)$, such that Eq. (3) can be rewritten as

$$\underset{\tilde{x},\tilde{z},x,z}{\text{minimize}} \quad f(\tilde{x}) + \mathcal{I}_{\{(u,v)\,:\,Au+v=b\}}(\tilde{x}, \tilde{z}) + \sum_{i=1}^{M} \mu_i \operatorname{dist}_{K_i}(z_i), \quad \text{subject to } (\tilde{x}, \tilde{z}) = (x, z). \tag{6}$$

Applying ADMM yields Algorithm 1; the full derivation is given in Appendix B. Line 2 is the equality-constrained QP of OSQP and COSMO, and it is solved by the same direct or indirect method (Appendix B.1). We prefer the indirect method, which consists of running preconditioned conjugate gradient (PCG) on the reduced system, because a learned policy can change ρ at every iteration and this requires a new factorization of the direct system each time. The only difference from COSMO is Line 4: COSMO projects onto K_i, while FlexCone applies the prox of the distance penalty, which has the closed forms in Table 1 along with the following properties.

Proposition 2.6 (FlexCone properties). *For each block i, let $\kappa_i = \mu_i/\rho$ and $C_i := \kappa_i B_i^* \cap K_i^\circ$.*

(a) $\operatorname{prox}_{\kappa_i \operatorname{dist}_{K_i}}(v_i) = v_i - \Pi_{C_i}(v_i)$.

(b) $y_i^{k+1} = -\rho\, \Pi_{C_i}(v_i^k)$ for every $k \geq 0$, thus $y^{k+1} \in D_\mu$. This holds for any ρ and α, including ones that change between iterations, and any $\tilde{x}^{k+1}$.

(c) If $\left\|\Pi_{K_i^\circ}(v_i)\right\|_* \leq \kappa_i$, then $\operatorname{prox}_{\kappa_i \operatorname{dist}_{K_i}}(v_i) = \Pi_{K_i}(v_i)$.

Algorithm 1: FlexCone

Input: problem data (P, q, A, b, K), penalties μ, step sizes $\sigma, \rho > 0$, relaxation $\alpha \in (0, 2)$, initial (x^0, z^0, y^0)

1 **for** $k = 0, 1, 2, \ldots$ *until converged* **do**

2 Solve $(P + \sigma I + \rho A^\top A)\tilde{x}^{k+1} = \sigma x^k - q + \rho A^\top (b - z^k - \rho^{-1} y^k)$; $\tilde{z}^{k+1} = b - A\tilde{x}^{k+1}$;

3 $x^{k+1} = \alpha \tilde{x}^{k+1} + (1-\alpha) x^k$; $\hat{z}^{k+1} = \alpha \tilde{z}^{k+1} + (1-\alpha) z^k$, $v^k = \hat{z}^{k+1} - \rho^{-1} y^k$;

4 $z_i^{k+1} = \mathrm{prox}_{(\mu_i/\rho)\,\mathrm{dist}_{K_i}}(v_i^k)$ for $i = 1, \ldots, M$; // COSMO: $\Pi_{K_i}(v_i^k)$

5 $y^{k+1} = y^k + \rho(z^{k+1} - \hat{z}^{k+1})$;

Part (a) shows that the prox can be evaluated through a projection onto a truncated polar cone. For the penalties in Table 1, this costs no more than a projection onto K_i (Appendix B.2). Part (b) is a key property for the learned policies developed in Section 2.4 and states that the dual iterates are bounded by μ at every iteration, whether the problem is feasible or not. Part (c) makes precise when FlexCone's iterates coincide with COSMO's.

Corollary 2.7 (Convergence). *Let $\sigma, \rho > 0$ and $\alpha \in (0, 2)$ be fixed, and suppose Eq. (3) has a minimizer. Then $(x^k, z^k, y^k) \to (\hat{x}, b - A\hat{x}, \hat{y})$, where $(\hat{x}, \hat{y})$ satisfies Eq. (5). If, in addition, $\|\hat{y}_i\|_* < \mu_i$, then there exists k_0 such that for all $k \geq k_0$ the step on block i coincides with COSMO's.*

This shows that a single termination criterion based on the residuals of Eq. (5) applies to both feasible and infeasible instances alike. Theorem 2.3b then decides whether the returned point solves Eq. (1). With an adaptive ρ, such as the residual-balancing rule used by OSQP/COSMO or our learned policies, Corollary 2.7 applies once ρ stops changing.

Table 1: Closed-form prox operators and their exactness condition on the penalty parameter (Theorem 2.3a, applied per block, where y^* denotes the corresponding block of the optimal multiplier).

Block	Penalty	Prox	Exactness
equality ($K = \{0\}$)	ℓ_1	two-sided soft-threshold	$\mu > \|y^*\|_\infty$
inequality ($K = \mathbb{R}^m_+$)	ℓ_1	one-sided soft-threshold of negative entries	$\mu > \|y^*\|_\infty$
SOC ($K = \mathcal{Q}^{m+1}$)	ℓ_2	project onto $\mathcal{Q}^{m+1}$, then shrink the residual	$\mu > \|y^*\|_2$
PSD ($K = \mathbb{S}^m_+$)	Frobenius	uniform shrinkage of the negative eigenvalues	$\mu > \|Y^*\|_F$
PSD ($K = \mathbb{S}^m_+$)	nuclear	one-sided threshold of the negative eigenvalues	$\mu > \|Y^*\|_{\mathrm{op}}$

2.4 Learned Hyperparameter Policies

FlexCone contains the same hyperparameters as COSMO, namely, a step size ρ for every constraint row and a relaxation parameter α. The best setting varies from problem to problem and from one iteration to the next. We therefore learn a policy that decides them, following the deep unfolding framework of Oshin et al. (2026b). At every iteration, a small neural network reads the current iterate and the residuals of Algorithm 1 and outputs the step sizes and relaxation for the next iteration, while a second network predicts a warm start (x^0, z^0, y^0) from the problem data. Both networks act row by row with shared weights, so one policy can be deployed on problems of any size. The step size predicted for a second-order or positive semidefinite block is averaged over that block because the prox (Line 4) couples its rows together. The untrained policy is initialized to reproduce a fixed ρ and $\alpha = 1.6$, and learns corrections that accelerate convergence. The networks are trained by unrolling Algorithm 1 for a fixed number of iterations K on a set of training problems and backpropagating a loss on the iterates through the entire unrolled solver, using the matrix-free and eigenvalue-stable gradient rules of Oshin et al. (2026b) for the linear solve and the prox.

Two key properties of FlexCone are relevant here. First, by Proposition 2.6c, as long as a block's multiplier stays below its penalty, the prox and its gradient coincide with the projection, so on feasible problems learned FlexCone is trained on the same signal as learned COSMO and inherits its gradient rules unchanged. Second, by Proposition 2.6b, the dual iterates are bounded by μ on *any* problem, whereas COSMO's grow without bound on infeasible ones. Any loss or policy feature that

depends on the dual iterates is therefore ill-conditioned for COSMO on exactly the instances where a solver needs the most help, which FlexCone avoids.

Training loss. Deep unfolding of a conic solver is usually trained by comparing the iterates against a precomputed solution x^*, which does not exist for an infeasible problem. We instead adopt a self-supervised loss built on the penalized optimality conditions Eq. (5), which is a stable criterion whether the problem is feasible or not. Let true slack be $s^k = b - Ax^k$ (rather than z^k, which need not equal $b - Ax^k$ before convergence). The residuals of iterate k are given by

$$r^k_{\text{stat}} = \left\|Px^k + q + A^\top y^k\right\|_\infty, \quad r^k_{\text{cone}} = \max_i \operatorname{dist}_{K_i}(s^k_i), \quad r^k_{\text{comp}} = \max_i \frac{\left|\langle s^k_i, y^k_i\rangle + \mu_i \operatorname{dist}_{K_i}(s^k_i)\right|}{\mu_i \max\{\|b\|_\infty, 1\}}.$$

Dual feasibility $y^k \in D_\mu$ holds at every FlexCone iterate by Proposition 2.6b, so it contributes nothing to the loss. The complementarity term is normalized by the penalty and the scale of b so that all three are measured in the units the solver's termination test uses, and $R^k := \max\left\{r^k_{\text{stat}}, r^k_{\text{comp}}, r^k_{\text{cone}}\right\} = 0$ exactly when (x^k, y^k) satisfies Eq. (5). The loss thus penalizes the solver for being above the desired tolerance:

$$L = \sum_{k=0}^{K} \operatorname{clip}\left(\frac{\log_{10}(R^k/\tau_{\text{low}})}{\log_{10}(\tau_{\text{high}}/\tau_{\text{low}})}, 0, 1\right), \tag{7}$$

where τ_{low} is one decade ($10\times$) below the tolerance the solver is deployed at, and τ_{high} is above the largest residual the untrained policy produces. The sum is therefore a smooth count of the iterations spent above tolerance: iterates below τ_{low} contribute no gradient, so the policy is not rewarded for accuracy beyond the deployed tolerance, and iterates above τ_{high} contribute a constant, so one diverging iterate cannot dominate the sum. The loss needs no labeled solutions, applies to feasible and infeasible instances alike, and cannot be exploited by pushing z^k away from the cone, since the cone term utilizes the true slack. Full training details are given in Appendix C.

3 Applications

3.1 Learning Benchmarks

Table 2: Conic programming benchmarks measuring convergence to 10^{-3} tolerance.

	COSMO		FlexCone		Learned COSMO		Learned FlexCone	
Problem Class	Iters	Time (s)	Iters	Time (s)	Iters	Time (s)	Iters	Time (s)
Max-cut	862 / 1098	1.76 / 2.24	735 / 1185	1.60 / 2.57	92 / **101**	**0.22** / **0.24**	**91** / 109	0.23 / 0.28
Lovász ϑ	107 / 119	0.32 / 0.35	156 / 184	0.48 / 0.57	400 / 672	1.32 / 2.23	**81** / **92**	**0.28** / **0.32**
Robust MPC	61 / 70	2.31 / 2.65	60 / 73	2.24 / 2.69	22 / 26	**0.46** / **0.56**	**18** / **22**	0.49 / 0.59
Robust KF	209 / 259	10.9 / 13.3	176 / 190	4.13 / 4.45	39 / 41	2.44 / **2.51**	**36** / **37**	**2.35** / 2.52
CS Double Integrator	304 / 685	32.4 / 66.8	1068 / 1199	19.3 / 21.1	30 / 33	3.03 / 3.22	**22** / **26**	**1.66** / **1.73**
CS Unicycle	10000 / 10000	600 / 600	6731 / 10000	114 / 600	42 / **47**	1.84 / **1.91**	**22** / 264	**0.70** / 19.0

We start by evaluating FlexCone and the learned policies on the benchmark suite of cone problems from Oshin et al. (2026b): max-cut and Lovász ϑ SDPs on random graphs, robust MPC and robust Kalman filtering SOCPs, and covariance steering (CS) SDPs that arise as SCP subproblems for a double integrator and a unicycle. Every instance is feasible, so this experiment isolates the solver and the learned policies from FlexCone's infeasibility handling. We compare COSMO and FlexCone with the residual-balancing step-size rule of OSQP against learned COSMO, using the checkpoints of Oshin et al. (2026b), and learned FlexCone, trained with the loss of Section 2.4 under the same protocol on 100 instances per class, with each block's penalty μ set to $1.5\times$ the largest multiplier norm observed on the training instances. Every solver must reach 10^{-3} on the stationarity residual, the ADMM primal residual, and the distance of the slack $b - Ax$ to the cone, within a cap of 10000 iterations or 600 s. Table 2 reports the median and 90th percentile over 20 held-out instances. Learned FlexCone matches or improves on learned COSMO in median iterations on every class and in median time on four of six classes, and requires 3–$10\times$ fewer iterations than the unlearned solvers on the SOCPs and CS problems. On the Lovász ϑ SDPs, where the unlearned solvers already converge in roughly 100 iterations, learned COSMO is almost $4\times$ slower than its own baseline,

whereas learned FlexCone improves on it. On the unicycle instances, the learned policies bring runs that exhausted the budget down to just tens of iterations. The protocol and additional statistics, including a variant trained with the supervised loss of Oshin et al. (2026b), are given in Appendix C.

3.2 Nonlinear Trajectory Optimization and Model Predictive Control

Table 3: Closed-loop iterative covariance steering. Success means reaching the goal while avoiding collisions. Statistics are mean $\pm$ std of successes over 20 scenarios $\times$ 3 noise seeds.

Solver	Successes	Collisions	ADMM Iters	Time (s)
COSMO	1/60	48	83.3k	106
FlexCone	52/60	7	$58.2\text{k} \pm 19.0\text{k}$	84 ± 23

We first consider closed-loop iterative covariance steering (iCS) of a unicycle through a field of three cylindrical obstacles (Ridderhof et al., 2019; Ridderhof & Tsiotras, 2022). At every $0.1\,\text{s}$ control step, a model-predictive controller linearizes the dynamics about its previous plan and solves one conic subproblem over a 25-step horizon. This subproblem solves for the nominal controls and disturbance feedback gains subject to a 78×78 linear matrix inequality (LMI) that bounds the planned state covariance, along with chance-constrained obstacle avoidance and speed limits modeled as second-order cones, plus a trust region constraint. Linearizing around a narrow gap between obstacles frequently makes the subproblem infeasible, even though the task itself can be completed. COSMO and FlexCone use identical hyperparameters, tolerances, and a 5000-iteration budget. When COSMO certifies a subproblem infeasible, or either solver exhausts its budget, the controller falls back to its previous plan. Otherwise, it applies the solver's returned point, which for FlexCone is the minimum-violation point on infeasible subproblems. On 20 held-out scenarios with 3 noise seeds each (Table 3), COSMO certifies 69% of all subproblems infeasible and completes only 1 of 60 episodes, typically becoming trapped within the first obstacle it approaches, unable to escape (Fig. 1, left). FlexCone gracefully handles infeasibilities and completes 52 of 60 episodes while using $3.2\times$ fewer ADMM iterations per episode. The task, solver settings, and additional results are provided in Appendix D.

Table 4: Closed-loop powered descent under a thrust constraint (80%) and acceleration disturbances ($0.1\,\text{m/s}^2$): 20 held-out scenarios, 3 disturbance seeds each. Unconverged/certified-infeasible steps fall back to the previous plan. We report median / 90th percentile over all episodes/subproblems.

	Closed-Loop Episode			Conic Subproblem	
Solver	Successes	Fallbacks	Success Time (s)	ADMM Iterations	Time (s)
COSMO	19/60 (32%)	9.5 / 19.1	9.7 / 30.4	240 / 5000	0.127 / 2.11
FlexCone	**54/60 (90%)**	**0 / 3**	**8.9 / 12.4**	**146 / 565**	**0.095 / 0.284**

We consider a 3-DoF powered descent guidance (PDG) problem based on Malyuta et al. (2022). The task is a minimum-fuel Mars landing problem with a free final time modeled using time-dilated dynamics. The vehicle starts 2–4 km away from the pad and must land at the origin with zero velocity. MPC solves one second-order cone program per control step. The plan keeps thrust at or below 80% of the maximum operating capacity, while the vehicle is allowed to use up to 100% to account for random acceleration disturbances that prevent achieving the nominal plan. We compare COSMO and FlexCone under the same fixed hyperparameters, iteration budget, tolerances, and fallback rule as above. On 20 held-out scenarios with 3 disturbance seeds each (Table 4), COSMO falls back on roughly a fifth of the control steps of a typical episode, mostly by exhausting its iteration budget, and lands 19 of 60 episodes, while FlexCone lands 54 of 60 with a median of zero fallbacks per episode. The dynamics, conic subproblem, scenario generation and solver settings are detailed in Appendix E.1.

Finally, we consider open-loop 6-DoF powered descent trajectory optimization (Szmuk & Acikmese, 2018). Each scenario is solved offline by SCP from the standard straight-line initialization, with the final time fixed at 1.6 times the minimum-time bound. At this horizon the first linearization violates the tilt or gimbal constraints on most scenarios, even though the nonlinear

Table 5: 6-DoF powered descent trajectory optimization via SCP. Each solver uses its best tuning and runs until 10^{-4} convergence. Success means the outer nonlinear task was solved to 5×10^{-3} tolerance. Statistics are mean $\pm$ std of successful solves over 20 held-out scenarios.

Solver	Successes	Violation ($\times 10^{-3}$)	SCP Iters	ADMM Iters	Time (s)
COSMO	9/20	0.79 ± 0.49	17.8 ± 3.3	3384 ± 997	22.2 ± 6.0
COSMO (virtual controls)	20/20	1.00 ± 0.48	19.4 ± 5.1	37271 ± 11522	404 ± 122
FlexCone	20/20	0.88 ± 0.44	19.6 ± 4.5	7423 ± 3733	45.7 ± 21.1

problem is feasible. This artificial infeasibility is conventionally handled by a *virtual control* approach, which adds penalized slack variables to the linearized dynamics (Mao et al., 2016; Szmuk & Acikmese, 2018). We compare COSMO, COSMO with virtual controls (COSMO-VC), and FlexCone, each within an identical SCP loop (Table 5). COSMO fails over half the scenarios because it halts wherever it encounters an infeasible subproblem, whereas COSMO-VC and FlexCone solve every scenario. Since FlexCone avoids modifying the subproblem, it uses $5\times$ fewer ADMM iterations and $9\times$ less wall-clock time than COSMO-VC, whose slacks add 1400 variables and 1400 constraint rows to every subproblem. The full formulation and protocol are given in Appendix E.2.

3.3 Classifier Robustness Certification

Finally, we use FlexCone to certify ReLU classifiers as robust, meaning that no small perturbation of an input changes the predicted class. Here an infeasible problem is the desired outcome rather than a failure. To certify one input, we ask, for each of the $C-1$ wrong classes, whether some perturbed input could make that class score at least as high as the true class. Each question is modeled as a linear feasibility problem, all $C-1$ of them share the same constraint matrix, and, when the network is robust, most of them are infeasible. The application therefore provides a natural mixed stream of feasible and infeasible problems that batch efficiently, and on infeasible problems the minimum-violation point (Proposition 2.5) identifies exactly the constraints that would have to be loosened for class j to win. The formulation, setup, and additional studies are given in Appendix F.

Table 6 reports verdicts on an adversarially trained MNIST MLP and CIFAR-10 CNN. FlexCone returns zero wrong verdicts against the exact references and reaches infeasible verdicts within 25 iterations on both datasets. On MNIST, FlexCone is over $50\times$ faster than HiGHS and $100\times$ faster than COSMO.jl solving the problems serially; on CIFAR-10 the margins are $4\times$ and $30\times$.

Table 6: Robustness certification via LP relaxation on adversarially trained MNIST and CIFAR-10 classifiers, mean $\pm$ standard deviation over three held-out folds. Infeasible is the fraction of problems whose infeasibility rules out a competing class. Time is per input over all $C-1 = 9$ problems; FlexCone solves each fold as one GPU batch, COSMO.jl and HiGHS (Huangfu & Hall, 2018) run serially on the CPU. Violation is the weighted ℓ_1 violation of the returned point on infeasible problems divided by the exact minimum (1 is optimal), and rows counts the constraints violated by more than 10^{-5}. Fold sizes, problem sizes, and the protocol are given in Appendix F.

			Time per input (s)			Violation ratio		Violated rows	
Dataset	ϵ	Infeasible	COSMO	FlexCone	HiGHS	COSMO	FlexCone	COSMO	FlexCone
MNIST	0.05	99.6%	0.52 ± 0.01	$\mathbf{0.0038 \pm 0.0001}$	0.31 ± 0.00	2.57 ± 0.01	$\mathbf{1.00 \pm 0.00}$	1048 ± 1	$\mathbf{16.7 \pm 0.6}$
MNIST	0.1	94.1%	1.00 ± 0.04	$\mathbf{0.0053 \pm 0.0003}$	0.36 ± 0.00	2.60 ± 0.01	$\mathbf{1.00 \pm 0.00}$	1037 ± 1	$\mathbf{14.3 \pm 0.6}$
CIFAR-10	1/255	98.3%	39.6 ± 1.1	$\mathbf{1.37 \pm 0.38}$	5.30 ± 0.13	7.15 ± 0.14	$\mathbf{1.00 \pm 0.00}$	10900 ± 57	$\mathbf{3.0 \pm 0.0}$
CIFAR-10	2/255	95.0%	48.3 ± 1.2	$\mathbf{1.52 \pm 0.27}$	6.21 ± 0.07	7.28 ± 0.17	$\mathbf{1.00 \pm 0.00}$	10900 ± 90	$\mathbf{4.3 \pm 0.6}$

Table 6 also compares the final COSMO and FlexCone iterates on the infeasible problems. Both are measured against the smallest possible weighted ℓ_1 violation, computed exactly by solving a ground-truth LP with HiGHS. COSMO stalls around $2.6\times$ (MNIST) and $7.3\times$ (CIFAR-10) the smallest violation, spread over about half of all constraints. Meanwhile, FlexCone reaches the smallest possible violation, while violating only a few constraints. These violated constraints show which pixel bounds and which ReLU relaxations would have to be loosened for class j to win, an interpretable output that projection-based splitting does not provide.

4 Related Work

Conic solvers and infeasibility detection. Interior-point methods are the standard choice for small- to medium-sized conic programs at high accuracy (Karmarkar, 1984; Goulart & Chen, 2024), but each iteration factorizes a linear system whose cost becomes a bottleneck at scale (Juditsky & Nemirovski, 2011). First-order solvers such as SCS (O'Donoghue et al., 2016), COSMO (Garstka et al., 2021), OSQP (Stellato et al., 2020), and PDLP (Applegate et al., 2021) replace the factorization with matrix-vector products and projections, warm-start easily, and have GPU implementations (Schubiger et al., 2020; Lu & Yang, 2025; Chen et al., 2026), although batching and differentiation through the iterations are not generally supported. All of these solvers detect infeasibility, either through a homogeneous self-dual embedding (O'Donoghue et al., 2016) or, for ADMM, through the successive differences of the iterates, which converge to a certificate (Banjac et al., 2019). In both cases the dual iterates diverge along the certificate, so the solver stops and discards the primal iterate. FlexCone instead converges to a KKT point of the penalized problem Eq. (3) with multipliers bounded by μ, so one termination test covers both cases.

Exact penalties and elastic formulations. The ℓ_1 penalty is exact for linear constraints once its weight exceeds $\|y^*\|_\infty$ (Pietrzykowski, 1969; Han & Mangasarian, 1979). Sℓ_1QP builds an SQP method on it (Fletcher, 1985), SNOPT switches to an ℓ_1 elastic mode when a QP subproblem is infeasible (Gill et al., 2005), filter methods enter a feasibility-restoration phase (Fletcher & Leyffer, 2002), and Byrd et al. (2010) steer the penalty weight so that SQP converges to a KKT point or a stationary point of the violation. In trajectory optimization, successive convexification adds ℓ_1-penalized virtual controls and buffers to the linearized constraints (Mao et al., 2016; Malyuta et al., 2022), which the practitioner must choose, weight, and tune by hand. FlexCone extends the exactness condition from linear constraints to arbitrary multi-block cones with the dual norm of each block's penalty (Table 1), applies the penalty inside the solver through a prox that costs no more than a projection rather than by enlarging the problem, and keeps the relaxation active on every instance, which is what allows batching and unrolling.

Learning to optimize. Learned solvers replace hand-designed parts of an iterative algorithm with components trained across instances (Chen et al., 2022; Amos, 2023). These include step sizes (Ichnowski et al., 2021; Sambharya & Stellato, 2024; Saravanos et al., 2025; Oshin et al., 2026a) and warm starts (Sambharya et al., 2023; 2024; Xiong et al., 2025) learned by deep unfolding (Monga et al., 2021), as well as optimization proxies (Van Hentenryck, 2025) and dual predictors (Tanneau & Van Hentenryck, 2024). Differentiable optimization layers (Amos & Kolter, 2017; Agrawal et al., 2019) differentiate the solution with respect to the problem data rather than through the solver trajectory, and so give no gradient for convergence speed. FlexCone builds on Deep FlexQP (Oshin et al., 2026a), which unfolds an ℓ_1 relaxation of polyhedral QPs, and on the batched conic unfolding of COSMO by Oshin et al. (2026b), extending the exact relaxation to second-order and semidefinite cones. All of these learned solvers train only on feasible instances, since a hard-constrained solver's diverging multipliers result in no finite target on infeasible problems. FlexCone's bounded multipliers enable the self-supervised loss of Section 2.4 to apply to both cases.

5 Conclusion

We introduced FlexCone, a first-order conic solver that unifies feasible and infeasible problems through an exact relaxation so they share one algorithm, one termination test, and one differentiable code path. This integration lets the solver batch on a GPU and learn from every instance, and across convex benchmarks, closed-loop SCP, and robustness certification it matched projection-based solvers when they succeed and returned a usable minimum-violation step when they do not. Our work generalizes and unifies many existing ideas across optimization and related fields, including ℓ_1-penalty formulations and virtual controls. Since penalized subproblems arise whenever a nonlinear program is convexified, FlexCone is a natural inner solver for SCP methods for optimal power flow, contact-rich robot planning, and constrained learning. Moreover, its bounded multipliers enable learning the penalty weights μ themselves and certifying infeasibility of outer nonlinear problems from the violated blocks it identifies.

AI USE STATEMENT

We used generative AI tools to help with code implementation and debugging, orchestrating experiments, and paper edits (wording, formatting, and spacing). These tools helped generate the datasets of problems studied in this work and assisted with data analysis. We have not used generative AI to develop new theoretical models, formulate mathematical claims, provide ingredients for mathematical claims, propose/refine hypotheses, or interpret results; all ideas are our own. The rest of the required disclosure tasks are not applicable to this work. We have thoroughly reviewed all AI-assisted work, including both writing and code. For example, we always verified any LLM-generated code via human reviewers along with extensive testing ensuring correctness of inputs and outputs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

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*, volume 32. Curran Associates, Inc., 2019.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023. ISSN 1935-8237. doi: 10.1561/2200000102.

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*, Sydney, Australia, 2017. PMLR.

David Applegate, Mateo Diaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan ODonoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20243–20257. Curran Associates, Inc., 2021.

Goran Banjac, Paul Goulart, Bartolomeo Stellato, and Stephen Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. *Journal of Optimization Theory and Applications*, 183(2):490–519, 2019. ISSN 0022-3239, 1573-2878. doi: 10.1007/s10957-019-01575-y.

Heinz H. Bauschke and Patrick L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. CMS Books in Mathematics. Springer New York, New York, NY, 2011. ISBN 978-1-4419-9466-0 978-1-4419-9467-7. doi: 10.1007/978-1-4419-9467-7.

Richard H. Byrd, Frank E. Curtis, and Jorge Nocedal. Infeasibility detection and SQP methods for nonlinear optimization. *SIAM Journal on Optimization*, 20(5):2281–2299, 2010. doi: 10.1137/080738222.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

Yuwen Chen, Danny Tse, Parth Nobel, Paul Goulart, and Stephen Boyd. CuClarabel: GPU acceleration for a conic optimization solver. *ACM Trans. Math. Softw.*, 52(3), 2026. ISSN 0098-3500. doi: 10.1145/3815420.

John W. Chinneck. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118 of *International Series in Operations Research & Management Science*. Springer, New York, NY, USA, 2008. ISBN 978-0-387-74932-7. doi: 10.1007/978-0-387-74932-7.

Jonathan Eckstein and Dimitri P. Bertsekas. On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318, 1992. ISSN 1436-4646. doi: 10.1007/BF01581204.

Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis*, pp. 269–286, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68167-2.

Roger Fletcher. An ℓ_1 penalty method for nonlinear constraints. In *Numerical Optimization 1984*, pp. 26–40. SIAM, Philadelphia, PA, 1985.

Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239–269, 2002. ISSN 1436-4646. doi: 10.1007/s101070100244.

Michael Garstka, Mark Cannon, and Paul Goulart. COSMO: A conic operator splitting method for convex conic problems. *Journal of Optimization Theory and Applications*, 190(3):779–810, 2021. ISSN 1573-2878. doi: 10.1007/s10957-021-01896-x.

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. ISSN 0036-1445, 1095-7200. doi: 10.1137/S0036144504446096.

Paul J. Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives, 2024.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models, 2018.

S. P. Han and O. L. Mangasarian. Exact penalty functions in nonlinear programming. *Mathematical Programming*, 17(1):251–269, 1979. ISSN 1436-4646. doi: 10.1007/BF01588250.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. ISSN 1867-2957. doi: 10.1007/s12532-017-0130-5.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21043–21055. Curran Associates, Inc., 2021.

Anatoli Juditsky and Arkadi Nemirovski. First-Order Methods for Nonsmooth Convex Large-Scale Optimization, I: General Purpose Methods. In *Optimization for Machine Learning*, pp. 0. The MIT Press, 2011. ISBN 978-0-262-29877-3. doi: 10.7551/mitpress/8996.003.0007.

N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, Stoc '84, pp. 302–311, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0-89791-133-4. doi: 10.1145/800057.808695.

Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *Operations Research*, 73(6):3440–3452, 2025. ISSN 0030-364X. doi: 10.1287/opre.2024.1069.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Danylo Malyuta, Taylor P. Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behçet Açıkmeşe. Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently. *IEEE Control Systems*, 42 (5):40–113, 2022. ISSN 1066-033X, 1941-000X. doi: 10.1109/MCS.2022.3187542.

Yuanqi Mao, Michael Szmuk, and Behcet Acikmese. Successive convexification of non-convex optimal control problems and its convergence properties. In *2016 IEEE 55th Conference on Decision and Control*, pp. 3636–3641, Las Vegas, NV, USA, 2016. IEEE. ISBN 978-1-5090-1837-6. doi: 10.1109/CDC.2016.7798816.

Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021. ISSN 1053-5888, 1558-0792. doi: 10.1109/MSP.2020.3016905.

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

NVIDIA Corporation. NVIDIA cuDSS: A high-performance cuda library for direct sparse solvers, 2026. URL https://docs.nvidia.com/cuda/cudss/index.html. Accessed: July 26, 2026.

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016. ISSN 1573-2878. doi: 10.1007/s10957-016-0892-3.

Alex Oshin, Rahul Vodeb Ghosh, Augustinos D. Saravanos, and Evangelos A. Theodorou. Deep FlexQP: Accelerated nonlinear programming via deep unfolding. In *The Fourteenth International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026a. doi: 10.48550/arXiv.2512.01565.

Alex Oshin, Rahul Vodeb Ghosh, and Evangelos A. Theodorou. Scalable deep unfolding of conic optimizers. In *Proceedings of the 10th Conference on Robot Learning*. PMLR, 2026b. doi: 10.48550/arXiv.2606.13825.

Tomasz Pietrzykowski. An exact potential method for constrained maxima. *SIAM Journal on Numerical Analysis*, 6(2):299–304, 1969. ISSN 0036-1429, 1095-7170. doi: 10.1137/0706028.

Jack Ridderhof and Panagiotis Tsiotras. Chance-constrained covariance steering in a gaussian random field via successive convex programming. *Journal of Guidance, Control, and Dynamics*, 45 (4):599–610, 2022. ISSN 1533-3884. doi: 10.2514/1.G005941.

Jack Ridderhof, Kazuhide Okamoto, and Panagiotis Tsiotras. Nonlinear uncertainty control with iterative covariance steering. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 3484–3490, Nice, France, 2019. IEEE. ISBN 978-1-7281-1398-2. doi: 10.1109/CDC40024.2019.9029993.

R. Tyrrell Rockafellar. *Convex Analysis*. Number 28 in Princeton Mathematical Series. Princeton University Press, Princeton, N.J, 1970. ISBN 978-0-691-08069-7.

Rajiv Sambharya and Bartolomeo Stellato. Learning algorithm hyperparameters for fast parametric convex optimization, 2024.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Proceedings of the 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pp. 220–234. PMLR, 2023.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

Augustinos D Saravanos, Hunter Kuperman, Alex Oshin, Arshiya Taj Abdul, Vincent Pacelli, and Evangelos Theodorou. Deep distributed optimization for large-scale quadratic programming. In *The Thirteenth International Conference on Learning Representations*, 2025.

Michel Schubiger, Goran Banjac, and John Lygeros. GPU acceleration of ADMM for large-scale quadratic programming. *Journal of Parallel and Distributed Computing*, 144:55–67, 2020. ISSN 0743-7315. doi: 10.1016/j.jpdc.2020.05.021.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020. ISSN 1867-2957. doi: 10.1007/s12532-020-00179-2.

Michael Szmuk and Behcet Acikmese. Successive convexification for 6-DoF Mars rocket powered landing with free-final-time. In *2018 AIAA Guidance, Navigation, and Control Conference*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, 2018. doi: 10.2514/6.2018-0617.

Mathieu Tanneau and Pascal Van Hentenryck. Dual Lagrangian learning for conic optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 55538–55561. Curran Associates, Inc., 2024. doi: 10.52202/079017-1764.

Pascal Van Hentenryck. Optimization learning, 2025.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5286–5295. PMLR, 2018.

Jinxin Xiong, Xi Gao, Linxin Yang, Jiang Xue, Xiaodong Luo, and Akang Wang. Solving quadratic programs via deep unrolled Douglas-Rachford splitting. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

A Proofs for the Exact Relaxation

Throughout the following proofs, let $s = b - Ax$ and $\hat{s} = b - A\hat{x}$. All statements are block-separable, so we drop the block index where no confusion arises for notational compactness.

Proof of Lemma 2.1. The distance function is the infimal convolution $\operatorname{dist}_K = (\|\cdot\| \,\square\, \mathcal{I}_K)$. The convex conjugate of an infimal convolution is the sum of the conjugates (Bauschke & Combettes, 2011), so

$$\operatorname{dist}_K^* = \mathcal{I}_{B^*} + \mathcal{I}_{K^\circ} = \mathcal{I}_{B^* \cap K^\circ}, \tag{8}$$

where we used the facts $\|\cdot\|^* = \mathcal{I}_{B^*}$ and $\mathcal{I}_K^* = \mathcal{I}_{K^\circ}$. Since K is non-empty, closed, and convex, dist_K is proper, convex, and continuous, so, by the Fenchel–Moreau theorem (Bauschke & Combettes, 2011), $\operatorname{dist}_K = (\mathcal{I}_{B^* \cap K^\circ})^*$, which proves the first identity in Eq. (4). The maximum is attained because the set $B^* \cap K^\circ$ is compact.

To prove the inequality, let $y \in K^*$ and $p \in K$. Then $\langle y, p \rangle \geq 0$, so $-\langle y, s \rangle \leq \langle y, p - s \rangle \leq \|y\|_* \|p - s\|$. Minimizing over $p \in K$ gives $-\langle y, s \rangle \leq \|y\|_* \operatorname{dist}_K(s)$.

Finally, applying the first identity with $u = -y/\mu_i$ and $K_i^\circ = -K_i^*$ yields $\mu_i \operatorname{dist}_{K_i}(s_i) = \max_{y_i \in D_{\mu_i}} \langle y_i, -s_i \rangle = \max_{y_i \in D_{\mu_i}} \langle y_i, A_i x - b_i \rangle$. Summing over blocks and adding $f(x)$ gives $\phi_\mu(x) = \max_{y \in D_\mu} \mathcal{L}(x, y)$. □

Proof of Proposition 2.2. By Lemma 2.1, $\mu_i \operatorname{dist}_{K_i} = \sigma_{C_i}$ is the support function of the compact convex set $C_i := \mu_i B_i^* \cap K_i^\circ$. Its subdifferential is the set of maximizers $\partial \sigma_{C_i}(s_i) = \arg\max_{u \in C_i} \langle u, s_i \rangle$ (Rockafellar, 1970). Since ϕ_μ is convex and finite everywhere, $\hat{x}$ minimizes it if and only if $0 \in \partial \phi_\mu(\hat{x})$, and applying the chain rule yields

$$\partial \phi_\mu(\hat{x}) = P\hat{x} + q - A^\top \textstyle\prod_i \partial \sigma_{C_i}(\hat{s}_i).$$

Hence, $\hat{x}$ is a minimizer if and only if there exists $u_i \in C_i$ with $\langle u_i, \hat{s}_i \rangle = \mu_i \operatorname{dist}_{K_i}(\hat{s}_i)$ and $P\hat{x} + q - A^\top u = 0$. Setting $\hat{y} = -u$ gives $\hat{y}_i \in -C_i = D_{\mu_i}$ and $\langle \hat{y}_i, A_i \hat{x} - b_i \rangle = \langle u_i, \hat{s}_i \rangle$, thus showing Eq. (5).

For saturation, combine the last condition in Eq. (5) with the inequality in Eq. (4), resulting in:

$$\mu_i \operatorname{dist}_{K_i}(\hat{s}_i) = -\langle \hat{y}_i, \hat{s}_i \rangle \leq \|\hat{y}_i\|_* \operatorname{dist}_{K_i}(\hat{s}_i). \tag{9}$$

If $\operatorname{dist}_{K_i}(\hat{s}_i) > 0$, then $\mu_i \leq \|\hat{y}_i\|_*$, and since $\hat{y}_i \in D_{\mu_i}$, this yields the reverse inequality. □

Proposition A.1 (Existence of a minimizer). *Eq. (3) has a minimizer if $P \succ 0$ or, more generally, if $q^\top d + \sum_i \mu_i \operatorname{dist}_{K_i}(-A_i d) > 0$ for every $d \neq 0$ with $Pd = 0$.*

If μ is small relative to q along a direction in which the violation grows only linearly, ϕ_μ is unbounded below, so the condition cannot be dropped.

Proof of Proposition A.1. We show that ϕ_μ (closed proper convex) has no nonzero direction of recession, which implies that it attains its minimum (Rockafellar, 1970). Fix x and $d \neq 0$. If $Pd \neq 0$, then $\phi_\mu(x + \tau d)$ must grow quadratically in τ, because the penalty is Lipschitz and grows at most linearly. If $Pd = 0$, then $f(x + \tau d) = f(x) + \tau q^\top d$. Since the cone distance $\operatorname{dist}_{K_i}$ is 1-Lipschitz and positively homogeneous because K_i is a cone, we have that

$$|\operatorname{dist}_{K_i}(s_i - \tau A_i d) - \tau \operatorname{dist}_{K_i}(-A_i d)| \leq \|s_i\|.$$

Hence $\lim_{\tau\to\infty} \phi_\mu(x + \tau d)/\tau = q^\top d + \sum_i \mu_i \operatorname{dist}_{K_i}(-A_i d)$, which is positive by assumption, so d is not a direction of recession. If $P \succ 0$, the condition holds vacuously. □

Proof of Theorem 2.3. Part (a). We first show that x^* minimizes ϕ_μ whenever $\mu_i \geq \|y_i^*\|_*$ for all i. Since $s^* \in K$ and $y^* \in K^*$, every term of $\langle y^*, s^* \rangle = \sum_i \langle y_i^*, s_i^* \rangle = 0$ is nonnegative, so each is zero. Therefore $\langle y_i^*, A_i x^* - b_i \rangle = 0 = \mu_i \operatorname{dist}_{K_i}(s_i^*)$, and $y^* \in D_\mu$, so (x^*, y^*) satisfies Eq. (5) and x^* minimizes ϕ_μ by Proposition 2.2, with $\min \phi_\mu = \phi_\mu(x^*) = f(x^*)$. Every solution x' of Eq. (1) is feasible with $f(x') = f(x^*)$, so $\phi_\mu(x') = f(x^*)$ and x' also minimizes ϕ_μ.

Conversely, let $\mu_i > \|y_i^*\|_*$ for all i and let $\hat{x}$ minimize ϕ_μ. By the inequality in Eq. (4), the convexity of $\mathcal{L}(\cdot, y^*)$, and stationarity, we have

$$f(\hat{x}) + \sum_i \|y_i^*\|_* \operatorname{dist}_{K_i}(\hat{s}_i) \geq f(\hat{x}) + \sum_i \langle y_i^*, A_i \hat{x} - b_i \rangle = \mathcal{L}(\hat{x}, y^*) \geq \mathcal{L}(x^*, y^*) = f(x^*).$$

Since also $\phi_\mu(\hat{x}) \leq \phi_\mu(x^*) = f(x^*)$, subtracting yields

$$\sum_i (\mu_i - \|y_i^*\|_*) \operatorname{dist}_{K_i}(\hat{s}_i) \leq 0.$$

Every term in the sum is nonnegative and every coefficient is positive, so $\operatorname{dist}_{K_i}(\hat{s}_i) = 0$ for all i, i.e., $\hat{x}$ is feasible. Thus, $f(\hat{x}) = \phi_\mu(\hat{x}) \leq f(x^*)$, so $\hat{x}$ solves Eq. (1).

Part (b). If $\|\hat{y}_i\|_* < \mu_i$, Eq. (9) forces $\operatorname{dist}_{K_i}(\hat{s}_i) = 0$, so $\hat{s}_i \in K_i$ and $\langle \hat{y}_i, \hat{s}_i \rangle = -\mu_i \operatorname{dist}_{K_i}(\hat{s}_i) = 0$. If this holds for all blocks, then $\hat{s} \in K$, $\hat{y} \in K^*$, $\langle \hat{y}, \hat{s} \rangle = 0$, and $P\hat{x} + q + A^\top \hat{y} = 0$, which are the KKT conditions Eq. (2). □

The strict inequality in Theorem 2.3a cannot be weakened: for $\min x$ subject to $x \geq 0$, we have $y^* = 1$, and $\mu = 1$ gives $\phi_\mu(x) = \max\{x, 0\}$, which is also minimized by every infeasible $x < 0$.

Proof of Corollary 2.4. If no block were saturated, Theorem 2.3b would imply that $\hat{x}$ is feasible, but this contradicts the assumption that Eq. (1) is infeasible. □

Proof of Proposition 2.5. Let $t_0 \leq t < t'$. Optimality of x_t and $x_{t'}$ gives the two inequalities:

$$f(x_t) + tV(x_t) \leq f(x_{t'}) + tV(x_{t'}), \qquad f(x_{t'}) + t'V(x_{t'}) \leq f(x_t) + t'V(x_t).$$

Adding them gives $(t' - t)(V(x_{t'}) - V(x_t)) \leq 0$, so $V(x_t)$ is nonincreasing, and the first inequality then gives $f(x_t) \leq f(x_{t'})$. For any x, optimality of x_t and monotonicity of $f(x_t)$ give $V(x_t) \leq V(x) + (f(x) - f(x_{t_0}))/t$, so $\limsup_t V(x_t) \leq V(x)$. Taking the infimum over x shows $V(x_t) \to V_{\min}$. Then, for any x' with $V(x') = V_{\min}$, optimality of x_t gives $f(x_t) \leq f(x') + t(V(x') - V(x_t)) \leq f(x')$. If $\bar{x}$ is a limit point of $\{x_t\}$, continuity gives $V(\bar{x}) = V_{\min}$ and $f(\bar{x}) \leq f(x')$, so $\bar{x}$ solves $\min_x f(x)$ subject to $V(x) = V_{\min}$. □

B Algorithm Derivation

We attach the multipliers w to $x - \tilde{x} = 0$ and y to the constraint $z - \tilde{z} = 0$. This is the opposite convention of OSQP and COSMO, but it ensures y matches the sign of the KKT multiplier in Eq. (2). With the other convention, $y^k \in -D_\mu$. Applying ADMM to problem Eq. (6) yields the updates

$$(\tilde{x}^{k+1}, \tilde{z}^{k+1}) = \underset{(\tilde{x},\tilde{z}):A\tilde{x}+\tilde{z}=b}{\arg\min} \tfrac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top\tilde{x} + \tfrac{\sigma}{2}\left\|\tilde{x} - x^k\right\|_2^2 + \tfrac{\rho}{2}\left\|\tilde{z} - z^k - \rho^{-1}y^k\right\|_2^2, \tag{10a}$$

$$x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k, \tag{10b}$$

$$\begin{aligned} z^{k+1} &= \arg\min_z \sum_{i=1}^M \mu_i \operatorname{dist}_{K_i}(z_i) + \tfrac{\rho}{2}\left\|z - \hat{z}^{k+1} + \rho^{-1}y^k\right\|_2^2 \\ &= \operatorname{prox}_{(\mu/\rho)\operatorname{dist}_K}(\hat{z}^{k+1} - \rho^{-1}y^k), \end{aligned} \tag{10c}$$

$$y^{k+1} = y^k + \rho(z^{k+1} - \hat{z}^{k+1}), \tag{10d}$$

where $\hat{z}^{k+1} = \alpha\tilde{z}^{k+1} + (1-\alpha)z^k$ and $\alpha \in (0, 2)$ is the relaxation parameter. The prox in Eq. (10c) separates over blocks. Note that we have dropped the multiplier w: since x is unconstrained in the second block, its update is $x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k - \sigma^{-1}w^k$, so the w update is $w^{k+1} = w^k + \sigma(x^{k+1} - \alpha\tilde{x}^{k+1} - (1-\alpha)x^k) = 0$ for all $k \geq 0$. Initializing with $w^0 = 0$ removes w from the algorithm entirely.

B.1 First Block: Linear System Solve

Let $\tilde{\nu}$ be the multiplier of the constraint $A\tilde{x} + \tilde{z} = b$ in Eq. (10a). The optimality conditions are

$$(P + \sigma I)\tilde{x} + q - \sigma x^k + A^\top\tilde{\nu} = 0, \quad \tilde{\nu} + \rho\tilde{z} - \rho z^k - y^k = 0, \quad A\tilde{x} + \tilde{z} = b.$$

Eliminating $\tilde{z} = z^k + \rho^{-1}(y^k - \tilde{\nu})$ yields the quasi-definite system

$$\begin{bmatrix} P + \sigma I & A^\top \\ A & -\rho^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\nu} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ b - z^k - \rho^{-1}y^k \end{bmatrix}, \tag{11}$$

which can be solved with a sparse $LDL^\top$ factorization. This is known as the *direct method* in OSQP (Stellato et al., 2020). Alternatively, eliminating $\tilde{\nu} = \rho(A\tilde{x} - b + z^k) + y^k$ yields the positive definite system

$$(P + \sigma I + \rho A^\top A)\tilde{x} = \sigma x^k - q + \rho A^\top(b - z^k - \rho^{-1}y^k),$$

which can be solved with the conjugate gradient method, the *indirect method* in OSQP. The slack variables are then given by $\tilde{z} = b - A\tilde{x}$.

B.2 Second Block: Prox Operators

Let $\kappa = \mu_i/\rho$ and let v be the input to the prox of block i. By Proposition 2.6a, $\operatorname{prox}_{\kappa\operatorname{dist}_K}(v) = v - \Pi_C(v)$ with $C = \kappa B^* \cap K^\circ$, so each case reduces to a projection onto C. The cone K determines K° and the choice of penalty determines B^*.

Zero cone, ℓ_1 norm. $K^\circ = \mathbb{R}^m$ and B^* is the ℓ_∞ ball, so $C = [-\kappa, \kappa]^m$ and the prox is the two-sided soft-threshold $\operatorname{sign}(v)\,(|v| - \kappa)_+$.

Nonnegative orthant, ℓ_1 norm. $K^\circ = \mathbb{R}^m_-$, so $C = [-\kappa, 0]^m$ and the prox is $v - \operatorname{clip}(v, -\kappa, 0)$, which shrinks only the negative entries toward zero.

Euclidean penalties (ℓ_2, Frobenius). When the penalty norm is the one induced by the quadratic term of the prox, the norm is self-dual and the prox is a cone projection followed by a shrinkage of the residual $r := v - \Pi_K(v)$:

$$\operatorname{prox}_{\kappa\operatorname{dist}_K}(v) = \Pi_K(v) + \left(1 - \frac{\kappa}{\|r\|_2}\right)_+ r, \tag{12}$$

with the convention that the shrinkage factor is 0 when $r = 0$. It returns exactly $\Pi_K(v)$ whenever $\operatorname{dist}_K(v) = \|r\|_2 \leq \kappa$, and costs no more than Π_K.

Second-order cone, ℓ_2 norm. With $\mathcal{Q}^{m+1} := \{(t,u) \in \mathbb{R} \times \mathbb{R}^m \mid \|u\|_2 \le t\}$, the projection is

$$\Pi_{\mathcal{Q}^{m+1}}(t,u) = \begin{cases} (t,u), & \|u\|_2 \le t, \\ (0,0), & \|u\|_2 \le -t, \\ \frac{t+\|u\|_2}{2}\left(1, \frac{u}{\|u\|_2}\right), & \text{otherwise}, \end{cases}$$

and substituting into Eq. (12) gives the prox.

PSD cone, Frobenius norm. Let $V = U\,\mathrm{diag}(\lambda)U^\top$ and split $\lambda = \lambda_+ + \lambda_-$ into positive and negative parts. Then $\Pi_{\mathbb{S}^m_+}(V) = U\,\mathrm{diag}(\lambda_+)U^\top$, the residual is $R = U\,\mathrm{diag}(\lambda_-)U^\top$ with $\|R\|_F = \|\lambda_-\|_2$, and Eq. (12) shrinks the negative eigenvalues uniformly:

$$\mathrm{prox}_{\kappa\,\mathrm{dist}_K}(V) = U\,\mathrm{diag}\left(\lambda_+ + \left(1 - \tfrac{\kappa}{\|\lambda_-\|_2}\right)_+ \lambda_-\right) U^\top.$$

PSD cone, nuclear norm. The nuclear norm $\|V\|_{\mathrm{nuc}} = \|\lambda(V)\|_1$ has the spectral norm as its dual, so $C = \{Y \mid -\kappa I \preceq Y \preceq 0\}$ and Π_C clips each eigenvalue to $[-\kappa, 0]$. The prox is the matrix analogue of the nonnegative-orthant case, $\hat{\lambda} = \lambda - \mathrm{clip}(\lambda, -\kappa, 0)$ entrywise, with U unchanged.

B.3 Proofs for the Algorithm

Proof of Proposition 2.6. Part (a). By Lemma 2.1, $\kappa\,\mathrm{dist}_K$ is the support function of $C = \kappa B^* \cap K^\circ$, so its conjugate is $\mathcal{I}_C$. The Moreau decomposition $v = \mathrm{prox}_g(v) + \mathrm{prox}_{g^*}(v)$ with $g = \kappa\,\mathrm{dist}_K$ and $\mathrm{prox}_{\mathcal{I}_C} = \Pi_C$ shows the claimed equation.

Part (b). With $v^k = \hat{z}^{k+1} - \rho^{-1}y^k$, Eq. (10d) is $y^{k+1} = \rho(z^{k+1} - v^k)$, and part (a) gives $z_i^{k+1} - v_i^k = -\Pi_{C_i}(v_i^k)$. Hence $y_i^{k+1} \in -\rho C_i = \mu_i B_i^* \cap K_i^* = D_{\mu_i}$. The argument uses only the last three lines of the ADMM loop defined in Algorithm 1, so it holds for any $\tilde{x}^{k+1}$ and any ρ, α at iteration k.

Part (c). If $\Pi_{K^\circ}(v) \in C$, then the nearest point of K° lies in C and is also the nearest point of C, i.e., $\Pi_C(v) = \Pi_{K^\circ}(v)$. The Moreau decomposition $v = \Pi_K(v) + \Pi_{K^\circ}(v)$ along with part (a) gives $\mathrm{prox}_{\kappa\,\mathrm{dist}_K}(v) = \Pi_K(v)$. COSMO's step with the same sign convention is $z^{k+1} = \Pi_K(v^k)$ and $y^{k+1} = \rho(z^{k+1} - v^k)$, so the two steps coincide. If $\|\Pi_{K^\circ}(v)\|_* < \kappa$, continuity of Π_{K° means the inequality holds within a neighborhood of v, on which the two maps are equal. □

Proof of Corollary 2.7. Let $(\hat{x}, \hat{y})$ satisfy Eq. (5), which exists by Proposition 2.2. Then the iterate $(\tilde{x}, \tilde{z}, x, z) = (\hat{x}, \hat{s}, \hat{x}, \hat{s})$ with multipliers $(w, y) = (0, \hat{y})$ is a saddle point of the Lagrangian of the split problem, since the first-block conditions are $P\hat{x} + q + A^\top\hat{y} = 0$ and the second-block condition is $-\hat{y}_i \in \partial(\mu_i\,\mathrm{dist}_{K_i})(\hat{s}_i)$, which is the last condition of Eq. (5). ADMM with over-relaxation $\alpha \in (0,2)$ applied to a convex problem whose Lagrangian has a saddle point converges to a saddle point (Eckstein & Bertsekas, 1992). Replacing the indicator of K by the proper closed convex penalty does not affect the argument. Every limit is a saddle point, so it satisfies Eq. (5).

For the second claim, $v^k = \hat{z}^{k+1} - \rho^{-1}y^k \to \hat{s} - \rho^{-1}\hat{y}$. If $\|\hat{y}_i\|_* < \mu_i$, Eq. (9) gives $\hat{s}_i \in K_i$ and $\langle \hat{y}_i, \hat{s}_i \rangle = 0$, and $-\rho^{-1}\hat{y}_i \in K_i^\circ$. By the Moreau decomposition, $\Pi_{K_i^\circ}(\hat{s}_i - \rho^{-1}\hat{y}_i) = -\rho^{-1}\hat{y}_i$, whose dual norm is strictly less than κ_i. By continuity of $\Pi_{K_i^\circ}$, the same strict inequality holds at v_i^k for all k large enough, and applying Proposition 2.6c completes the proof. □

C Learned Benchmark Details

This appendix gives the details behind the conic programming benchmarks presented in Section 3 (Table 2).

C.1 Problem Classes

We use six of the problem classes of Oshin et al. (2026b); the details on the problem formulations are provided in their appendix. Table 7 lists their sizes, including the number of variables and

constraints, and the structure of the cone blocks. Max-cut and Lovász ϑ are SDP relaxations on Erdős–Rényi graphs $G(n, 0.5)$ with $n = 50$ and $n = 60$ nodes, each with one PSD block and one block of equalities. Robust MPC is a chance-constrained regulation problem for a chain of three masses over a 50-step horizon, and robust Kalman filtering is a Huber-loss state estimation problem over 100 time steps; both are SOCPs with many small second-order cones. The two covariance steering (CS) classes are the SDP subproblems of an SCP loop on a double integrator and a unicycle over 30 steps, with an LMI on the terminal covariance, a trust region, and a set of dynamics equality blocks. For the double integrator, each instance is the first linearization of a randomly sampled start and goal, while for the unicycle, each sampled trajectory yields the ten subproblems of its SCP loop, so the 120 instances come from 12 separate trajectories. Every class has 120 instances with reference solutions computed by an interior-point solver (Clarabel) to a tolerance of 10^{-10}; the first 100 are used for training and the last 20 are held out for evaluation, the same split as in Oshin et al. (2026b). For the unicycle, the held-out instances are the subproblems of the last two trajectories.

Table 7: Sizes of the benchmark problem classes: decision variables n, constraint rows m, and cone blocks with their row counts in parentheses (eq. = equality block, for PSD blocks the matrix dimension is given).

Problem Class	n	m	Cone blocks
Max-cut ($n = 50$)	1275	1325	1 eq. (50), 1 PSD (50×50)
Lovász ϑ ($n = 60$)	1830	2685–2765	1 eq. (855–935), 1 PSD (60×60)
Robust MPC ($T = 50$)	507	1425	1 eq. (306), 1 orthant (912), 51 SOC (4–7)
Robust KF ($N = 100$)	1204	1200	1 eq. (700), 1 orthant (200), 100 SOC (3)
CS Double Integrator ($N = 30$)	3904	8363	32 eq. (4), 1 orthant (240), 61 SOC (3–5), 1 PSD (124×124)
CS Unicycle ($N = 30$)	2943	4861	32 eq. (3), 1 orthant (180), 61 SOC (3–4), 1 PSD (93×93)

C.2 Training Protocol

We follow the protocol of Oshin et al. (2026b) for every learned solver, so that the learned COSMO and learned FlexCone rows of Table 2 differ only in the solver being unrolled and the loss. The hyperparameter policy and the warm-start network are the ones described in that work: two 32×32 ReLU MLPs that act on every constraint row and predict additive updates to the logits of ρ and α from the row's iterate, multiplier and residual features, with ρ bounded to $[10^{-3}, 10^{3}]$ and α to $(0, 2)$ through a squashed sigmoid, and a 64×64 MLP that predicts the warm start elementwise from the problem data. The output layers are zero-initialized, so an untrained policy reproduces a fixed ρ and $\alpha = 1.6$ and an untrained warm start is the zero initialization. Both are trained end-to-end through the unrolled solver with Adam, a cosine learning-rate schedule from 10^{-4} to 10^{-5}, gradient-norm clipping at 1, and super-batches of 5 instances stacked into one block-diagonal problem. The linear system is solved by preconditioned conjugate gradient with a Jacobi preconditioner, and the PSD prox is differentiated with the eigenvalue-stable rule of Oshin et al. (2026b). All training runs use double precision on one RTX 4090.

The learned COSMO models are the checkpoints of Oshin et al. (2026b), trained for 100 epochs on the log of the primal and dual residuals along the unroll, with the checkpoint selected by the median final residual on validation instances. The learned FlexCone models are trained for up to 50 epochs with early stopping once the validation loss has not improved by 1% for 5 epochs, and the checkpoint with the lowest validation loss is kept. Table 8 lists the per-class settings. The unroll length K is the one used for learned COSMO on the same class. The starting step sizes are COSMO's defaults for the COSMO models, whereas for FlexCone they were chosen by a sweep of the untrained solver on the training instances, since in FlexCone ρ also sets the prox threshold μ/ρ and COSMO's values do not transfer. The penalty of each block is $\mu_i = 1.5 \max \|y_i^*\|_*$, the maximum taken over the reference multipliers of the 100 training instances in the block's own dual norm (Table 1). Note that a block whose multiplier is zero on every training instance is given the largest threshold of any block in the class. At this penalty, no block is saturated at any training solution, so by Theorem 2.3a the relaxation is exact on the training distribution.

We train two FlexCone models per class. The first uses the loss of Section 2.4 with $\tau_{\text{low}} = 10^{-4}$, $10\times$ below the 10^{-3} benchmark tolerance, and τ_{high} set to the power of ten above the largest residual produced by the untrained policy on three training instances. This is the model reported as learned FlexCone in Table 2. The second, which we call the parity model, uses the labeled loss of Oshin

et al. (2026b): the log of the primal and dual residuals along the unroll, normalized by their initial values, plus the distance of the iterate to the reference solution x^* normalized by its initial value. Its purpose is to check that switching the solver from COSMO to FlexCone does not by itself change what is learned. The two FlexCone models share every other setting, so the pair ablates the effect of the loss.

Table 8: Per-class training settings for the learned FlexCone models. Starting step sizes are given for the equality rows and for all other rows. Epochs are the number trained before early stopping for the parity and true-KKT (Eq. (7)) losses.

Problem Class	Unroll K	ρ (eq. / other)	τ_{high}	Epochs (parity / true-KKT)	PSD penalty
Max-cut	300	10 / 0.3	10^4	8 / 10	Frobenius
Lovász ϑ	100	1000 / 30	10^3	12 / 7	Frobenius
Robust MPC	300	100 / 0.1	10^3	16 / 41	–
Robust KF	100	100 / 3	10^2	25 / 33	–
CS Double Integrator	150	10 / 3	10^7	19 / 38	Frobenius
CS Unicycle	150	10 / 3	10^8	20 / 29	Frobenius

C.3 Benchmark Protocol

All experiments in this paper were run in PyTorch 2.13 on a machine with an Intel i9-13900K CPU and an NVIDIA RTX 4090 GPU with 24 GB of memory. All four solvers share one implementation that differs only in the second ADMM block (Section 2.3), and every run solves one instance at a time on the GPU in double precision. The unlearned COSMO and FlexCone rows use the residual-balancing rule of OSQP (Stellato et al., 2020), which rescales ρ by the ratio of the primal and dual residuals and adopts the new value only when it changes by more than a factor of $5\times$, starting from the same step sizes as the corresponding learned model and from a zero initialization. The linear system is solved by PCG to an absolute tolerance of 10^{-8} for every solver. A run is converged at the first iterate whose stationarity residual $\left\|Px + q + A^\top y\right\|_\infty$, ADMM primal residual $\|Ax + z - b\|_\infty$, and cone violation $\max_i \operatorname{dist}_{K_i}(b_i - A_i x)$ are all at most 10^{-3}. For COSMO, this last term is always true, but it is an extra requirement for FlexCone. Runs are limited to 10000 iterations and 600 s, and a run that reaches either budget is charged both caps in the statistics. Wall-clock time includes all setup time, including the learned warm start.

C.4 Parity Model Results

Table 9: Learned solvers on the same 20 held-out instances as Table 2: learned COSMO, learned FlexCone trained with the labeled loss of Oshin et al. (2026b) (parity), and learned FlexCone trained with Eq. (7) (true-KKT). Statistics are median / 90th percentile. Runs that reach the cap are charged 10000 iterations and 600 s.

	Learned COSMO		Learned FlexCone (parity)		Learned FlexCone (true-KKT)	
Problem Class	Iters	Time (s)	Iters	Time (s)	Iters	Time (s)
Max-cut	92 / 101	0.22 / 0.24	100 / 116	0.25 / 0.29	91 / 109	0.23 / 0.28
Lovász ϑ	400 / 672	1.32 / 2.23	176 / 268 (1 fail)	0.61 / 0.94	81 / 92	0.28 / 0.32
Robust MPC	22 / 26	0.46 / 0.56	32 / 45	0.68 / 0.94	18 / 22	0.49 / 0.59
Robust KF	39 / 41	2.44 / 2.51	38 / 39	3.00 / 3.30	36 / 37	2.35 / 2.52
CS Double Integrator	30 / 33	3.03 / 3.22	27 / 29	2.36 / 2.39	22 / 26	1.66 / 1.73
CS Unicycle	42 / 47	1.84 / 1.91	20 / 140	0.67 / 5.17	22 / 264	0.70 / 19.0

Table 9 reports the parity model alongside the two learned solvers of Table 2 on the same held-out instances. The parity model lands within a factor of $1.5\times$ learned COSMO in median iterations on every class, faster on four of the six (by $2.3\times$ on Lovász ϑ, with one failure) and slower on max-cut and robust MPC, so replacing the projection by the prox does not by itself change what deep unfolding learns. The loss of Section 2.4 then improves on the parity model on every class in median iterations, most clearly on robust MPC ($32 \to 18$) and Lovász ϑ ($176 \to 81$), without using any solution labels. The one weakness of both FlexCone models is the tail on the unicycle problems, where a few subproblems take several hundred iterations, which learned COSMO does not exhibit.

D Closed-Loop Iterative Covariance Steering Details

This appendix gives the details behind the closed-loop iterative covariance steering experiment presented in Section 3.2.

D.1 Task Description

The plant is a unicycle with state (p_x, p_y, θ) and control (v, ω), discretized with a time step of 0.1 s over an episode of $N = 100$ steps. The goal lies 8 m from the start. Its direction and the initial and goal headings are randomly sampled per scenario. The nominal speed is limited to $|v| \leq 1.4$ m/s, which forces the vehicle to use most of the episode to reach the goal. The plant is subject to additive Gaussian process noise with per-step standard deviations of $(0.0126, 0.0126, 0.0063)$, and the controller receives noisy measurements of the full state with standard deviations $(0.015, 0.015, 0.006)$, which it filters using an extended Kalman filter (EKF). Three cylindrical obstacles of radius 0.8 m are placed along the straight line from the start to goal positions. Their centers are randomly sampled to lie between 20% and 90% of the way to the goal, on alternating sides of the line to make the trajectories non-trivial.

Feasible scenarios were sampled by ensuring that an offline, full-horizon SCP solve succeeds on the obstacle-free problem. Feasibility with the obstacles is not certified offline, but the 52 successful FlexCone episodes show that the task can be completed in most scenarios. An episode is considered a success if the trajectory is collision-free and the true final state lies within $\sqrt{\operatorname{tr} \Sigma_N} = 0.173$ of the goal (norm taken over position in meters and heading in radians, with $\Sigma_N = 0.1^2 I$).

D.2 Conic Subproblem

At each MPC step t, the controller solves one convex subproblem over a window of $H = \min(25, N - t)$ steps, so the window shrinks over the final 25 steps of the episode. The dynamics are linearized about the previous plan, shifted forward by one step, and the initial state distribution is given by the current EKF estimate. The decision variables are the nominal controls and a causal disturbance-feedback policy, $u_i = \bar{v}_i + \sum_{j<i} K_{ij} w_j$, which makes the planned state covariance an affine function of the gains. The subproblem contains the following constraint blocks:

- **Initial state and dynamics** (zero cone): the linearized, time-varying dynamics of the planned mean.
- **Covariance bound** (a 78×78 positive semidefinite cone): an LMI requiring the planned covariance at the end of the window to be less than $\Sigma_N^{\text{cap}} + (N - t - H)W$, where $\Sigma_N^{\text{cap}} = (0.35 \times 0.1)^2 I$ and W is the per-step process-noise covariance. The bound tightens as the window approaches the end of the episode and equals Σ_N^{cap} once the window is at the final step, so the same subproblem formulation can be used at every step of the episode.
- **Obstacle avoidance** (second-order cones): one chance constraint per obstacle and window step with violation probability $\varepsilon = 0.01$. Each is the obstacle's keep-out region linearized as a half-space at the reference point, inflated by a 0.05 m buffer and backed off by $\Phi^{-1}(1 - \varepsilon) \approx 2.33$ standard deviations of the planned position along the half-space normal.
- **Speed limit** (second-order cones): $|\bar{v}_i| \leq 1.4$ m/s on the nominal controls.
- **Trust region** (second-order cones): bounding the deviation of the planned states and controls from the reference.

The cost is quadratic in the planned mean, centered on the goal, with control weights $R = \operatorname{diag}(0.1, 0.05)$ and the terminal state weight is scaled by a factor of 30. With the full 25-step window, a subproblem has $n = 1928$ variables, $m = 9313$ constraint rows and 178 cone blocks. The outer SCP loop is initialized using a straight line start from the initial state to the goal. Both solvers are warm started from the shifted solution from the previous step.

D.3 Solvers and Protocol

Both COSMO and FlexCone share one implementation that differs only in the second ADMM block (projection vs. distance prox, Section 2.3). We apply 10 iterations of Ruiz equilibration (Stel-

lato et al., 2020) before each solve, and run both on the CPU using a sparse $LDL^\top$ factorization (QDLDL). Both use a fixed step size $\rho = 1$ on the equality rows and $\rho = 0.03$ on all other rows, relaxation $\alpha = 1.6$, termination tolerances $\epsilon_{\text{abs}} = 10^{-3}$ and $\epsilon_{\text{rel}} = 10^{-4}$, and a budget of 5000 iterations per subproblem. COSMO's infeasibility certificate uses a tolerance of 10^{-5}. If a subproblem is certified infeasible or the solver budget is reached, the controller falls back to its previous plan shifted forward by one step.

The FlexCone penalty μ_i is set to 1.5 times the largest dual norm $\|y_i\|_*$ observed in exact solves of the subproblems along five calibration scenarios. The trust-region penalty is scaled by 10^6, so the trust region acts effectively like a hard constraint. We selected the hyperparameters using the calibration scenarios with two noise seeds each. For FlexCone, we swept the step size on the equality rows over $\{0.1, 0.3, 1, 3, 10, 30, 300\}$, the step size on the second-order cone and PSD rows separately, the relaxation $\alpha \in \{1.0, 1.3, 1.6, 1.8, 1.9\}$, and the penalty scale ($0.5\times$ and $2\times$), and chose the setting with the fewest budget-limited steps. For COSMO, we tried the same settings along with the adaptive ρ rule, but it only completed at most 1 of 10 episodes, so we use FlexCone's hyperparameters for both solvers. The 20 held-out scenarios were never used during development or tuning. Three noise seeds were tested per each scenario, which changes the initial state and the process and measurement noise (obstacles are fixed per scenario).

D.4 Supplementary Experimental Results

Table 10 reports all 60 episodes of each solver, rather than only the successful ones. For COSMO only 926 (15%) subproblems converge within the budget. The rest either hit the budget or certify infeasibility, and these steps fall back to the previous plan. In most episodes, the vehicle remains stuck inside the first obstacle, as in Fig. 1 (left), and can never escape.

Meanwhile, FlexCone never certifies infeasibility and its only fallbacks come from reaching the iteration budget (377 of 6000 steps). Its failures occur exclusively during these fallback scenarios. Notably, the 7 collisions occur across 7 different scenarios, so no single scenario is consistently hard for FlexCone. Across noise seeds 0/1/2, FlexCone completes 16/18/18 of the 20 scenarios vs. 1/0/0 for COSMO.

Table 10: Closed-loop iCS over all 60 episodes (20 held-out scenarios $\times$ 3 noise seeds). Success means reaching the goal without colliding. Fallbacks are counted per episode and split into certified-infeasible and budget-limited steps. Position error is the distance of the final position from the goal. Other statistics are median / 90th percentile over all episodes or all 6000 subproblems.

	Closed-Loop Episode							Conic Subproblem	
Solver	Successes	Reached Goal	Collisions	Fallbacks	Position Error (m)	ADMM Iterations	Time (s)	ADMM Iterations	Time (s)
COSMO	1/60	6/60	48	68.8 + 15.8	5.28 / 6.28	190k / 225k	231 / 272	1207 / 5000	1.37 / 5.78
FlexCone	52/60	59/60	7	0 + 6.3	0.029 / 0.068	59.0k / 93.4k	86 / 127	224 / 1469	0.28 / 1.80

E Powered Descent Guidance Details

This appendix describes the two powered descent guidance (PDG) experiments of Section 3.2: closed-loop model predictive control of a 3-DoF lander (Appendix E.1) and open-loop 6-DoF trajectory optimization (Appendix E.2).

E.1 3-DoF Closed-Loop Model Predictive Control

Vehicle and dynamics. The vehicle is the minimum-fuel Mars lander of Malyuta et al. (2022), with position $r \in \mathbb{R}^3$, velocity $v \in \mathbb{R}^3$, mass m and thrust $T \in \mathbb{R}^3$ in a planet-fixed frame:

$$\dot{r} = v, \qquad \dot{v} = g + \frac{T}{m} - \omega \times (\omega \times r) - 2\,\omega \times v, \qquad \dot{m} = -\alpha \|T\|_2, \tag{13}$$

where $g = (0, 0, -3.71)\,\text{m/s}^2$, ω is the rotation of Mars and $\alpha = 1/(I_{sp}\, g_e \cos\phi)$ with $I_{sp} = 225\,\text{s}$ and cant angle $\phi = 27°$. The thrust magnitude satisfies $\rho_{\min} \le \|T\|_2 \le \rho_{\max}$ with $\rho_{\min} = 4972\,\text{N}$

and $\rho_{\max} = 13258\,\mathrm{N}$, its direction is within $\gamma_p = 40^\circ$ of vertical, $\|T\|_2 \cos\gamma_p \le e_z^\top T$, the position satisfies the 86° glide-slope constraint $\cot(86^\circ)\max(|r_x|, |r_y|) \le r_z$, and the speed satisfies $\|v\|_2 \le 138.9\,\mathrm{m/s}$. The dry mass is $1505\,\mathrm{kg}$ and the vehicle carries $600\,\mathrm{kg}$ of propellant.

Scenarios. Initial states are sampled with altitude in $[600, 2800]\,\mathrm{m}$, horizontal distance up to $3.5\,\mathrm{km}$ and a downward velocity of 25–165 m/s. A state is kept only if the losslessly convexified fixed-final-time problem (Malyuta et al., 2022) is feasible for some final time, and the nominal final time t_f is set to the midpoint of its feasible window. We also discard a state if the controller's first subproblem, without disturbances, is infeasible, so that every scenario can be solved at its nominal final time. The first 20 accepted scenarios are used for calibration and the next 20 are used as held out test problems, and each is run with 3 different disturbance seeds.

Closed-loop control. The control period is $\Delta t_c = t_f/50 \approx 2\,\mathrm{s}$. The plant integrates Eq. (13) with 20 RK4 sub-steps per control period, holding the commanded thrust constant, and experiences an acceleration disturbance $a \sim \mathcal{N}(0, 0.1^2 I)\,\mathrm{m/s^2}$ each step. The actuator rotates any command outside the 40° pointing cone onto its boundary, clamps the magnitude to $[\rho_{\min}, \rho_{\max}]$ and caps it so that the step cannot burn below the dry mass. An episode ends when the planned remaining flight time falls to one control period, when the vehicle enters the touchdown box, or after 100 steps. It is considered a success if the final position is within 1 m of the pad, the final speed is at most $0.5\,\mathrm{m/s}$, and no path constraint is violated by more than 10^{-3} of its characteristic scale, i.e., about 3 m of glide-slope violation for a 3 km descent and $0.07\,\mathrm{m/s}$ of excess speed.

Conic subproblem. At each step the controller solves a free-final time optimal control problem over $N = 50$ intervals, where the remaining flight time σ is itself a decision variable. The first interval is fixed at the control period and the rest share the remaining time, $\Delta_0 = \Delta t_c$ and $\Delta_k(\sigma) = (\sigma - \Delta t_c)/(N-1)$ for $k \ge 1$, so the first planned control is exactly the one the plant executes. Writing $x_k = (r_k, v_k, m_k)$ and $\eta = 0.8$ for the planning thrust margin, the nonlinear problem at step t from the measured state x_t is

$$
\begin{aligned}
\min_{\sigma, x, T} \quad & \alpha \sum_{k=0}^{N-1} \Delta_k(\sigma)\|T_k\|_2 + \tfrac{w_\sigma}{2}(\sigma - \sigma_{\mathrm{nom}})^2 \\
\text{s.t.} \quad & x_{k+1} = F(x_k, T_k, \Delta_k(\sigma)), \quad x_0 = x_t, \quad r_N = 0, \quad v_N = 0, \quad m_N \ge m_{\mathrm{dry}}, \\
& \rho_{\min} \le \|T_k\|_2 \le \rho_{\max}, \quad \|T_k\|_2 \le \eta\rho_{\max}, \quad \|T_k\|_2 \cos\gamma_p \le e_z^\top T_k, \\
& \text{glide slope and speed limit at every node}, \quad \Delta t_c \le \sigma \le t_f^{\max},
\end{aligned}
\tag{14}
$$

where F is one RK4 step of Eq. (13), $t_f^{\max}$ is the upper end of the scenario's feasible window, and the proximal term with $w_\sigma = 25\,\mathrm{kg/s^2}$ keeps the planned touchdown near the nominal instant $\sigma_{\mathrm{nom}} = t_f - t\,\Delta t_c$. The planning margin requires the plan to use at most 80% of the maximum thrust, while the vehicle can deliver 100%, leaving headroom to reject disturbances. Near the end of a descent the disturbances can make this requirement unsatisfiable, thus causing the subproblem to become infeasible.

The convex subproblem linearizes Eq. (14) about the previous plan, shifted forward one step. The dynamics use the exact Jacobians of the RK4 map, the thrust lower bound is linearized along the reference thrust direction, the fuel term is linearized in σ, and the norms of T_k are written with second-order cone epigraphs. A trust region bounds the deviation of the plan from the reference by 2 in scaled units and σ by 1 s. Each subproblem is an SOCP with $n = 558$ variables and $m = 2640$ constraint rows. The controller is initialized using the optimum of the losslessly convexified problem, and every later solve is warm-started with the previous plan's solution recomputed onto the new window, since the remaining flight time has changed.

Solvers and protocol. COSMO and FlexCone share one implementation. Both apply 10 iterations of Ruiz equilibration and factor the KKT system on the CPU with QDLDL (Stellato et al., 2020). Both use a fixed step size of $\rho = 0.1$ (100 on equality rows), relaxation $\alpha = 1.6$, and a budget of 5000 iterations. Termination follows the OSQP convention on the unscaled residuals, with $\epsilon_{\mathrm{abs}} = 10^{-3}$ and $\epsilon_{\mathrm{rel}} = 10^{-6}$. COSMO's infeasibility tolerance is 10^{-4}. The step size and the choice between fixed ρ and the adaptive rule were selected on six calibration scenarios over $\rho \in \{0.01, 0.1, 1\}$. On this distribution of problems, neither the adaptive ρ rule nor the learned

policies of Section 2.4 gave a practical benefit over fixed hyperparameters, so both solvers run with the fixed settings above. The FlexCone penalty μ_i of each constraint family is 1.5 times the largest dual norm observed in exact solves on the 20 calibration scenarios without disturbances, and the trust-region penalty is scaled by 10^6. If a subproblem is certified infeasible or exhausts its budget, the controller discards the returned point and applies the next control of its previous plan.

E.2 6-DoF Open-Loop Trajectory Optimization

Vehicle and dynamics. We use Problem 1 of Szmuk & Acikmese (2018) in its normalized units, with state $x = (m, r, v, q, \omega_B) \in \mathbb{R}^{14}$ and body-frame thrust $T_B \in \mathbb{R}^3$ as the control:

$$\begin{aligned} \dot{m} &= -\alpha \|T_B\|_2, \qquad \dot{r} = v, \qquad \dot{v} = \frac{1}{m} C_{I/B}(q)\, T_B + g, \\ \dot{q} &= \tfrac{1}{2}\Omega(\omega_B)\, q, \qquad J_B \dot{\omega}_B = r_{T,B} \times T_B - \omega_B \times J_B \omega_B. \end{aligned} \tag{15}$$

Here q is the attitude quaternion, $C_{I/B}(q)$ the corresponding rotation matrix, $g = -e_1$ (the first axis points up), $J_B = 10^{-2} I$, the gimbal point is $r_{T,B} = -10^{-2} e_1$, and $I_{sp} = 30$ giving $\alpha = 1/30$. Every node must satisfy the dry-mass bound $m \geq 1.0$, the 20° glide slope $\tan(20^\circ)\|(r_2, r_3)\|_2 \leq r_1$, the 90° tilt limit $\cos(90^\circ) \leq 1 - 2(q_2^2 + q_3^2)$, the angular-rate limit $\|\omega_B\|_2 \leq 60^\circ$ per unit time, the thrust bounds $0.3 \leq \|T_B\|_2 \leq 5$, and the 20° gimbal limit $\|T_B\|_2 \cos(20^\circ) \leq e_1^\top T_B$. The vehicle starts with mass 2.0 and fixed position, velocity and angular rate, but a free attitude. It must end at the origin with velocity $-0.1\, e_1$, identity attitude, zero angular rate, and thrust along its body axis.

Problem formulation. We fix the final time and minimize fuel $m_0 - m_N$. The final time is 1.6 times a minimum-time bound, obtained from a translational double-integrator relaxation of each scenario. The trajectory is discretized over $N = 50$ intervals with one RK4 step per interval and its exact Jacobian. The nonlinear constraints are linearized about the current iterate, and the thrust lower bound along the reference thrust direction. Scenarios are sampled around the initial state of Szmuk & Acikmese (2018), and the held-out scenarios were never used for calibration. The SCP starts from the straight-line initialization given by Szmuk & Acikmese (2018). At this horizon, the first subproblem is infeasible on most scenarios, even though every scenario has a feasible solution. On the calibration scenarios, 58% of the nonlinearly feasible scenarios meet an infeasible subproblem, and the certificates attribute the infeasibility to the tilt or gimbal constraints.

SCP loop. Every solver runs inside the same SCP loop with an ℓ_1 merit function, a Han–Powell penalty weight, an acceptance ratio test, and a box trust region of initial radius 2.0 in scaled units. The loop stops after 60 iterations or when the step and the nonlinear constraint violation fall below 10^{-3} and 5×10^{-3}, respectively. FlexCone's penalty μ is the loop's current merit weight, with the trust-region rows scaled by 10^6, so FlexCone adds no hyperparameters. The virtual-control variant adds a slack $\nu_k \in \mathbb{R}^{14}$ to the linearized dynamics of every interval and the penalty $w_\nu \|\nu_k\|_1$ to the objective, with the ℓ_1 norm written in epigraph form. We set $w_\nu = 10$, the smallest weight on our calibration grid (10–10^4) that drives the slack to zero at convergence. All weights reach the same final fuel (within 0.1%). The slacks and their epigraph add 1400 variables and 1400 constraint rows to every subproblem.

Solvers and protocol. All three solvers apply 10 iterations of Ruiz equilibration with termination on the unscaled residuals at $\epsilon_{\text{abs}} = 10^{-3}$, with a fixed step size, relaxation $\alpha = 1.6$, and a budget of 10k iterations per subproblem. The step sizes were tuned per solver on the calibration scenarios. COSMO uses $\rho = 3$ (3000 on equality rows), COSMO with virtual controls (COSMO-VC) $\rho = 0.03$ (0.3), and FlexCone $\rho = 3$ (300). COSMO-VC is limited by its dual residual and favors a much smaller step size than the other two solvers. A subproblem that exhausts its budget passes its last iterate to the outer SCP loop as an inexact solve. An outcome is considered a success if the final trajectory, re-simulated with a fine RK4 integrator, violates no constraint or terminal condition by more than 5×10^{-3}.

F Classifier Robustness Certification Details

This appendix gives the details behind the robustness certification experiments introduced in Section 3.3.

F.1 Problem Formulation

We consider a network of affine layers (W_l, b_l), $l = 1, \ldots, L$, with a ReLU activation between consecutive layers. Let x_0 be an input with true label c_0. An adversary may change each pixel by at most ϵ (an ℓ_∞ perturbation) while keeping pixel values in $[0, 1]$, so the perturbed input x ranges over the box $[\ell^x, u^x] = [\max(x_0 - \epsilon, 0), \min(x_0 + \epsilon, 1)]$. Interval bound propagation (IBP) (Gowal et al., 2018) computes lower and upper bounds $\ell^{(l)} \le z^{(l)} \le u^{(l)}$ on every hidden neuron's pre-activation value. The bounds are exact for the first layer but become loose in later layers. We therefore use tighter backward (CROWN-style) bounds (Zhang et al., 2018) so the LP relaxation is valid. Given these bounds, each ReLU can be replaced with the standard triangle relaxation (Ehlers, 2017; Wong & Kolter, 2018), the smallest convex set that contains the graph of the ReLU on $[\ell_i, u_i]$. Writing $z^{(l)}$ for the pre-activation and $h^{(l)}$ for the relaxed post-activation of hidden layer l, with $h^{(0)} = x$, the LP for a competing class $j \neq c_0$ is given by

$$\underset{x,z,h}{\text{minimize}} \; m_j(h^{(L-1)}) \coloneqq \big(W_L[c_0, :] - W_L[j, :]\big)^\top h^{(L-1)} + \big(b_L[c_0] - b_L[j]\big) \tag{16a}$$

$$\text{subject to} \; z^{(l)} = W_l h^{(l-1)} + b_l, \;\; l = 1, \ldots, L-1, \tag{16b}$$

$$h_i^{(l)} = z_i^{(l)} \;\text{ if } \ell_i^{(l)} \ge 0, \;\; h_i^{(l)} = 0 \;\text{ if } u_i^{(l)} \le 0, \tag{16c}$$

$$h_i^{(l)} \ge 0, \;\; h_i^{(l)} \ge z_i^{(l)}, \;\; h_i^{(l)} \le \tfrac{u_i^{(l)}}{u_i^{(l)} - \ell_i^{(l)}} \big(z_i^{(l)} - \ell_i^{(l)}\big) \;\text{ if } \ell_i^{(l)} < 0 < u_i^{(l)}, \tag{16d}$$

$$\ell^x \le x \le u^x. \tag{16e}$$

The objective m_j is the margin by which the true class c_0 beats class j in logit space. If the smallest achievable margin is positive, no bounded input can make class j win, meaning that class j is ruled out. The input is certified robust when every competing class is ruled out. Since the LP relaxes the network, a certificate is always valid, but it may fail to certify some inputs that are in fact robust.

Eq. (16) always has at least one feasible point, which is the clean input together with its true activations. We instead solve an equivalent decision form of the LP. We add the margin constraint $m_j(h^{(L-1)}) \le 0$, which means that "class j scores at least as high as c_0", and ask if there exists a point that satisfies all constraints. Class j is therefore ruled out exactly when this system is infeasible. We call each pair of input and competing class an *instance* or *system*. The same formulation covers both MNIST and CIFAR-10. Convolutions are rewritten using their equivalent dense affine map, which means that convolutions can be modeled exactly like MLPs.

F.2 Networks and Training

We train classifiers on MNIST and CIFAR-10 that consist of plain chains of affine layers and RELU activations, without normalization, pooling, or skip connections, so Eq. (16) applies to them directly; see Table 11 for a description of each architecture. We chose the layer widths and the training parameters so that the LP can certify a meaningful fraction of inputs, rather than to maximize accuracy. For reference, on a standard MNIST MLP (97.7% test accuracy), a projected gradient descent (PGD) attack (Madry et al., 2018) succeeds on 68% of inputs at $\epsilon = 0.05$, and the LP certifies almost nothing at that radius. We add two changes to fix this: (1) adversarial training with PGD (8 steps, with the attack radius increased gradually over the first 5 epochs, and the checkpoint chosen by accuracy under attack on a validation set), and (2) an ℓ_1 penalty on the weights. On MNIST at $\epsilon = 0.05$, the fraction of inputs the LP certifies rises from 5% without the penalty to 30%, 80%, and 90% at penalty weights of 10^{-5}, 10^{-4}, and 3×10^{-4}, respectively. Sparser weights give tighter neuron bounds and therefore fewer unstable neurons, i.e., neurons whose sign is not fixed over the box and which must be replaced by the triangle relaxation. We use a penalty weight of 3×10^{-4} on MNIST and 10^{-4} on CIFAR-10.

F.3 Decision Certification

Since residual-based stopping criteria return inexact iterates, we certify the decisions through an alternative set of criteria. Let $\mathcal{B} = \{v \mid \ell^v \le v \le u^v\}$ be the box given by the propagated bounds on every variable. Every point that satisfies the relaxation lies in $\mathcal{B}$. An `infeasible` answer therefore requires a vector $\lambda \in K^*$ with $\lambda^\top b < \min_{v \in \mathcal{B}} (A^\top \lambda)^\top v$. Any such λ proves that no feasible point

Table 11: Classifier architectures and problem sizes. CIFAR-10 convolutions are 4×4 with stride 2. The number of constraints depends on how many neurons are unstable; we give the median at each radius (MNIST $\epsilon = 0.05$ / 0.1, CIFAR-10 $1/255$ / $2/255$).

Dataset	Architecture	Hidden ReLUs	Clean Accuracy	Accuracy under PGD	Variables n	Constraints m
MNIST	MLP 784–128–64–10	192	97.0%	87% ($\epsilon = 0.1$)	1,168	2,024 / 2,106
CIFAR-10	conv16–conv32–fc100–10	6,244	59.6%	47.2% ($\epsilon = 2/255$)	15,560	19,009 / 19,403

exists, and the solver's clipped dual iterate $y^k \in D_\mu$ serves as this proof. This is formalized through the following result.

Proposition F.1 (Bounded Farkas certificate). *Let $K = \{0\}^{m_{\text{eq}}} \times \mathbb{R}_+^{m_{\text{in}}}$ and $\lambda \in K^*$, i.e., $\lambda_i \geq 0$ on inequality rows and $\lambda_i = 0$ on free rows. If*

$$\lambda^\top b < \min_{v \in \mathcal{B}} (A^\top \lambda)^\top v = \sum_i \min\left\{(A^\top \lambda)_i \ell_i^v,\ (A^\top \lambda)_i u_i^v\right\}, \tag{17}$$

then $\{v \in \mathcal{B} \mid b - Av \in K\}$ is empty.

Proof. If $v \in \mathcal{B}$ satisfies $b - Av \in K$, then $\langle \lambda, b - Av \rangle \geq 0$ because $\lambda \in K^*$, so $\lambda^\top b \geq (A^\top \lambda)^\top v \geq \min_{\mathcal{B}} (A^\top \lambda)^\top v$, which contradicts Eq. (17). □

The standard Farkas certificate requires $A^\top \lambda = 0$ exactly, which an iterative solver only reaches in the limit. The test above allows the box bounds to absorb the portion of $A^\top \lambda$ that is not yet zero. It therefore holds for any $\lambda \in K^*$, and we apply it directly to the current dual iterate y^k, which lies in $D_\mu \subset K^*$ by Proposition 2.6b, restricted to the active rows and evaluated in `float64`.

For `feasible` answers, a tolerance check on the iterate can be inexact. Concretely, with a tolerance of e.g. 10^{-4} per constraint, it can declare some systems feasible that are actually infeasible, namely those whose optimal margin is positive but very small. We therefore repair the iterate towards a point that satisfies the relaxation exactly. We clip x into the input box, recompute every $z^{(l)}$ and $h^{(l)}$ by a forward pass through the network, clipping $h_i^{(l)}$ of each unstable neuron into $[\max(z_i^{(l)}, 0), \text{chord}]$, and then check the margin constraint at the resulting point. We tested this repair in a controlled study, in which the margin threshold is shifted to the exact LP optimum $\pm\delta$, for δ down to 10^{-2}, so that every system lies on one side of the feasibility boundary. Every answer was correct (1728 of 1728 answered, 0 wrong), and feasible answers arrived 2–6× faster than when waiting for the residuals to reach the tolerance. The number of iterations to certify an answer grows as $\approx \delta^{-0.6}$ on both sides of the boundary, and this exponent does not change for $\rho \in \{0.01, 0.03, 0.1\}$ or $\alpha \in \{1.0, 1.6, 1.9\}$. The repair time is included within the reported solve times.

F.4 Baselines and Solver Details

HiGHS (Huangfu & Hall, 2018, dual simplex method) solves the optimization form Eq. (16) directly using one LP per class pair and run serially on the CPU. This serves as the ground-truth oracle and we check the answers of each method against the sign of the optimal margin.

COSMO.jl (Garstka et al., 2021, v0.8.9) solves the decision problem serially on the CPU with its sparse $LDL^\top$ factorization, using tolerances 10^{-5} and at most 5000 iterations. We test three configurations: the default (adaptive ρ with Anderson acceleration), fixed ρ with Anderson acceleration, and fixed ρ without acceleration. We found that adaptive ρ and Anderson acceleration both hurt on this class of problems, so we use a fixed tuned ρ.

FlexCone We solve $G \times (C-1)$ systems in parallel, where G is the number of inputs and $C-1$ is the number of competing classes. We use zero padding so that all inputs have the same number of constraint rows. During setup, we apply 10 iterations of Ruiz equilibration following Stellato et al. (2020) so that constraints have comparable norms. For these feasibility problems, Line 2 of Algorithm 1 becomes $(\sigma I + \rho A^\top A)\tilde{x} = \sigma x^k + \rho A^\top (b - z^k - \rho^{-1} y^k)$. The coefficient matrix is therefore shared across all $C-1$ systems of each input and a matrix factorization can be reused at every iteration as long as ρ is fixed. For MNIST, we cache the explicit dense matrix inverse as the linear system is not too large, while for CIFAR-10, we instead solve the quasi-definite KKT system

Eq. (11) using cuDSS (NVIDIA Corporation, 2026). Moreover, certified systems are frozen within a batch once an answer is available, so the runtime depends largely on the slowest instance within each batch. We use a penalty of $\mu = 1$ when computing answers and $\mu = 0.1$ when computing the minimum-violation point, a fixed $\rho = 0.1$, $\sigma = 10^{-6}$, $\alpha = 1.6$. Answers are checked every 25 iterations, and we use a feasibility tolerance of 10^{-4} for the repaired point, a relative slack of 10^{-9} in the infeasibility test, and at most 1,000 iterations for answers and 3,000–5,000 for minimum-violation points. We deliberately switch off the residual-balancing rule that adapts ρ. With a zero objective, the dual residual is nearly zero from the start, so the rule tends to continuously increase ρ causing the primal residual to stall. The same problem breaks COSMO's default configuration on these systems.

Evaluation protocol. Each dataset is split into three disjoint held-out folds. MNIST uses 128 inputs per fold, giving $128 \times 9 = 1152$ feasibility problems per fold and 3456 per radius, and CIFAR-10 uses 64 inputs per fold, giving 576 problems per fold and 1728 per radius. All statistics in Table 6 are the mean $\pm$ standard deviation across the three folds. The infeasible fraction in Table 6 is the fraction of the $C - 1$ problems per input that are infeasible, i.e., the percentage of competing classes that are ruled out. Reported times are per input and cover all $C - 1 = 9$ problems of that input, including the repair step of the previous subsection. The minimum-violation columns of Table 6 are computed on the infeasible problems only, after at most 3000 FlexCone or COSMO iterations. The violation is the weighted ℓ_1 distance of the returned point to the constraints, divided by the smallest achievable weighted violation, which we compute exactly by solving a ground-truth LP with HiGHS, so a value of 1 is optimal. Violated rows counts the constraints violated by more than 10^{-5}; the total number of constraints per problem is the median m listed in Table 11.

F.5 Supplementary Experimental Results

Table 12 compares FlexCone with COSMO on the MNIST and CIFAR-10 networks at each radius and reports the number of iterations for feasible and infeasible decisions. Overall, COSMO requires nearly 5–15$\times$ more iterations to find infeasible certifiates than FlexCone at the median, and 5–24$\times$ more at the 90th percentile. On feasible systems, the difference is more modest, ranging from 1.5–6$\times$ at the median and 1.5–5.7$\times$ at the 90th percentile. Since answers are certified within a short number of iterations, we do not expect learned policies would help much on these classes. Indeed, we tried training the learned policies following the protocol in Section 2.4, but they hurt the total wall-clock time and did not improve any outcome. Therefore, we report results using FlexCone without learning.

Table 12: Answers on MNIST and CIFAR-10 at each radius on the three held-out folds, counts summed over folds. Iterations to the answer are given as median / 90th percentile, separately for infeasible and feasible systems. Time is per input, mean $\pm$ standard deviation across folds. FlexCone solves each fold in one GPU batch while COSMO.jl runs serially on the CPU.

Solver	Dataset	ϵ	Correct / Wrong / Unanswered	Infeasible Iterations	Feasible Iterations	Time (s)
COSMO.jl	MNIST	0.05	3456 / 0 / 0	121 / 294	54 / 267	0.524 ± 0.007
FlexCone	MNIST	0.05	3456 / 0 / 0	25 / 25	33 / 58	0.0038 ± 0.0001
COSMO.jl	MNIST	0.1	3446 / 0 / 10	388 / 1188	117 / 525	1.002 ± 0.036
FlexCone	MNIST	0.1	3456 / 0 / 0	25 / 50	50 / 92	0.0053 ± 0.0003
COSMO.jl	CIFAR-10	1 / 255	1728 / 0 / 0	121 / 228	133 / 275	39.6 ± 1.1
FlexCone	CIFAR-10	1 / 255	1728 / 0 / 0	25 / 42	75 / 183	1.37 ± 0.38
COSMO.jl	CIFAR-10	2 / 255	1728 / 0 / 0	188 / 601	321 / 692	48.3 ± 1.2
FlexCone	CIFAR-10	2 / 255	1728 / 0 / 0	25 / 50	50 / 183	1.52 ± 0.27