A Differentiable Interior-Point Method in Single Precision

Jon Arrizabalaga
MIT
Cambridge, MA 02139
jonarri@mit.edu

Kevin Tracy
Intrinsic, Google
Mountain View, CA 94043
kevintracy48@gmail.com

Zachary Manchester
MIT
Cambridge, MA 02139
zacm@mit.edu

Abstract

Primal-dual interior-point methods solve constrained convex optimization problems to tight tolerances with speed and robustness. Their solutions are also efficiently differentiable with respect to the problem data through the implicit function theorem. However, the standard treatment of primal-dual complementarity makes the underlying linear systems increasingly ill-conditioned near the solution. While this ill-conditioning is often benign in double precision, it can be catastrophic in single precision, preventing interior-point methods from fully exploiting the accelerated hardware that underpins modern machine learning. This paper introduces a differentiable interior-point method for convex quadratic programs designed for low-precision arithmetic. By using an alternative complementarity representation, we remove the complementarity-induced ill-conditioning of the underlying linear systems — even near the solution — a property that is essential for computing accurate gradients and avoiding arithmetic exceptions. As a result, our method enables interior-point solvers to reliably solve and differentiate optimization problems in single precision that were previously confined to double precision. We demonstrate the approach through an ablation study against the standard interior-point formulation and applications in bilevel and end-to-end learning settings where differentiating through constrained optimization is essential. The source code is available at `https://github.com/qpax-solver/qpax`.

1 Introduction

Constrained convex programs are central to control [1, 2, 3, 4, 5], simulation [6, 7, 8], operations research [9, 10, 11, 12, 13], and machine learning [14, 15, 16, 17, 18]. They provide a principled way to encode domain-specific structure, often leading to optimization problems that can be solved efficiently and reliably [19]. With the growth of large-scale data and compute, increasing attention has shifted toward making these optimization pipelines differentiable, so that problem parameters can be tuned directly from data [20, 21, 22]. While this does not replace expert modeling, it substantially expands the design space by allowing optimization-based components to be embedded within larger trainable systems [23, 24, 25].

In recent years, differentiable convex optimization has attracted considerable interest, particularly as a mechanism for integrating optimization layers into neural architectures [26, 27, 28, 29]. This line of work has led to practical modeling tools, such as CVXPY Layers, that make it straightforward to incorporate parameterized convex programs into standard machine learning workflows [30].

Primal-dual interior-point methods (IPMs) have long served as the algorithmic workhorse for solving these optimization problems [19]. Due to their rapid convergence, rigorous theoretical foundations,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

and practical reliability, they remain the method of choice when small primal-dual residuals and reliable derivatives are required [31, 32, 33, 34].

However, the robust performance of IPMs has historically required double-precision (f64) arithmetic to maintain the numerical stability of the linear systems solved at each iteration [35, 36]. This requirement fundamentally clashes with modern machine learning hardware and software stacks, which prioritize single-precision (f32) and mixed-precision computation for throughput and memory efficiency [37, 38]. While lower-precision formats accelerate parallelized pipelines, they severely amplify the numerical instabilities inherent in standard IPMs, rendering conventional solvers impractical for contemporary machine learning workflows [39].

Motivated by this critical bottleneck, we ask: *how can we enable reliable, differentiable interior-point methods in single precision?* As a foundational step toward this broader goal, we focus on convex Quadratic Programs (QPs). QPs provide a canonical setting for studying these numerical issues, and while the ideas developed here extend to other convex cones, such extensions are beyond the scope of the present work.

Contributions: We introduce a differentiable interior-point method for convex QPs that (i) natively supports low-precision arithmetic and (ii) yields higher-accuracy gradients than standard IPM formulations. To realize this, we make the following contributions:

1. A primal-dual interior-point algorithm whose linear systems are free of complementarity-induced ill-conditioning, avoiding the numerical instabilities affecting standard IPMs in low-precision arithmetic.
2. A relaxation-based differentiation pipeline that leverages this algorithm to extract unique, smooth, and highly accurate derivatives of the solution with respect to the problem data.
3. Extensive empirical validation on challenging benchmarks, demonstrating that our approach reliably solves and differentiates convex QPs in single precision where the standard formulation fails.

Related work. This paper builds on three lines of work. The first, *low-precision optimization*, accelerates interior-point methods with mixed-precision linear algebra, iterative refinement, and GPU implementations [40, 41, 42, 43, 44]. All of them keep the conventional complementarity condition, whose scaling $D(s \oslash z)$ still diverges as the barrier parameter vanishes; iterative refinement mitigates this divergence, whereas we remove its source, and the two can be combined. The second, *implicit complementarity*, replaces the complementarity condition by a nonlinear map: semismooth Newton methods on NCP functions [45, 46] target the exact condition, whereas log-domain [47], softplus [48], and Lie-group [49] parameterizations target the perturbed one smoothly. We adopt the softplus retraction of [48], introduced there for solving, not differentiating, QPs in double precision. The third, *differentiable optimization layers* [27, 28, 50, 51, 52], differentiates through convex programs via the implicit function theorem, in all cases in double precision; [29] introduced the central-path relaxation on which we build. We therefore claim none of these ingredients as new: our contribution is the end-to-end pipeline that solves and differentiates convex QPs in native single precision with the same implicit-complementarity KKT structure throughout.

2 Preliminaries

In this section, we introduce quadratic programming, the standard primal-dual interior-point framework, and its differentiability. This provides the foundations required to introduce our method.

2.1 Quadratic Programming

A convex quadratic program (QP) and its equivalent slack-variable formulation are given by

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^\top Qx + q^\top x \\ \text{subject to} \quad & Ax = b, \\ & Gx \le h, \end{aligned} \tag{1a}$$

$$\iff$$

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & \frac{1}{2}x^\top Qx + q^\top x \\ \text{subject to} \quad & Ax = b, \\ & Gx + s = h, \\ & s \ge 0. \end{aligned} \tag{1b}$$

Here, $x \in \mathbb{R}^n$ denotes the primal variable, the positive semidefinite $Q \in \mathbb{S}^n_+$ and $q \in \mathbb{R}^n$ define the quadratic objective, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ specify the equality constraints, $G \in \mathbb{R}^{p \times n}$ together with $h \in \mathbb{R}^p$ define the inequality constraints, and the slack variable $s \in \mathbb{R}^p$ converts inequalities into equalities.

Introducing Lagrange multipliers $y \in \mathbb{R}^m$ for the equality constraints and $z \in \mathbb{R}^p$ for the inequality constraints, the Lagrangian of the slack formulation is

$$\mathcal{L}(x, s, y, z) = \frac{1}{2} x^\top Q x + q^\top x + y^\top (Ax - b) + z^\top (Gx + s - h). \tag{2}$$

The Karush–Kuhn–Tucker (KKT) conditions for optimality are given by

$$\begin{aligned} r_t &\coloneqq \quad Qx + q + G^\top z + A^\top y = 0, && \text{(3a)} \\ r_c &\coloneqq \quad z \odot s = 0, && \text{(3b)} \\ r_i &\coloneqq \quad Gx + s - h = 0, && \text{(3c)} \\ r_e &\coloneqq \quad Ax - b = 0, && \text{(3d)} \\ & \quad\quad s \geq 0, && \text{(3e)} \\ & \quad\quad z \geq 0, && \text{(3f)} \end{aligned}$$

where $\odot$ denotes elementwise multiplication and r_t, r_c, r_e, r_i refer to the stationarity, complementarity, equality and inequality residuals. A primal–dual iterate (x^*, s^*, y^*, z^*) is globally optimal if it satisfies (3).

2.2 Primal-Dual Interior-Point Methods

To solve (3), standard primal-dual IPMs perturb the complementarity condition in (3b) as

$$r_{c,\kappa} \coloneqq z \odot s - \kappa, \tag{4}$$

where κ is the central path parameter. As $\kappa \to 0$, the solution of the perturbed system approaches that of (3). For a fixed value of κ, Newton's method is applied to the modified KKT conditions, leading to the linear system

$$\begin{bmatrix} Q & A^T & G^T & 0 \\ A & 0 & 0 & 0 \\ G & 0 & 0 & I \\ 0 & 0 & D(s) & D(z) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_t \\ r_e \\ r_i \\ r_{c,\kappa} \end{bmatrix}. \tag{5}$$

where $D(\cdot)$ denotes the diagonal matrix formed from a vector. The dominant computational cost at each iteration lies in solving this system, typically via a condensed formulation introduced later.

Once the search directions are computed, a fraction-to-boundary step-size rule is applied to preserve the positivity of (s, z). The maximum step size $\alpha_{\max} \leq 1$ such that $s + \alpha_{\max} \Delta s \geq 0$ and $z + \alpha_{\max} \Delta z \geq 0$ is given in closed form by

$$\alpha_{\max} = \min \left(1, \min_{i:\, \Delta s_i < 0} \frac{-s_i}{\Delta s_i}, \min_{i:\, \Delta z_i < 0} \frac{-z_i}{\Delta z_i} \right), \tag{6}$$

and the step is taken as $\alpha = \min(1, \eta\, \alpha_{\max})$ with $\eta = 0.99$. For further details on primal-dual interior-point methods and their implementation, see [53].

2.3 Differentiable Interior-Point Quadratic Programming

When differentiating numerical optimizers, it is computationally inefficient to propagate derivatives through the solver's internal iterations. An alternative approach is to solve the optimization problem nominally, and then compute the exact derivatives at the final solution using the implicit function theorem [54] (see Appendix A for specific details on its applicability to interior-point QPs). However, directly applying the implicit function theorem to differentiate a quadratic program can yield uninformative subgradients near active inequality constraints [28]. To address this issue, derivatives are not evaluated at the returned solution—a strictly interior iterate with a small $\kappa^* = s^\top z / p$—but instead at a relaxed point along the central path defined by $\kappa_{\text{relax}} > \kappa^*$, reached while preserving $s, z > 0$. This relaxation smooths the local geometry of the feasible set, producing gradients that remain well

behaved, even near sharp constraint boundaries [29]. Concretely, the backward pass returns the exact sensitivities $\partial x_{\kappa_{\text{relax}}}/\partial\theta$, with $\theta = (Q, q, A, b, G, h)$, of the perturbed problem

$$\underset{x,\, s>0}{\text{minimize}} \quad \frac{1}{2}x^\top Qx + q^\top x - \kappa_{\text{relax}} \sum\nolimits_i \log s_i \quad \text{subject to} \quad Ax = b, \quad Gx + s = h, \tag{7}$$

whose KKT conditions satisfy $s \odot z = \kappa_{\text{relax}}\mathbf{1}$, rather than derivatives of the nonsmooth solution map of the original QP. As a result, differentiating an interior-point QP naturally decomposes into three stages:

1. Solve the quadratic program to the desired convergence tolerance.
2. Relax the converged solution to a central path parameter κ_{relax}.
3. Compute the exact derivatives at the relaxed point.

The choice of κ_{relax} depends on the desired degree of smoothing. When only mild regularization is required, a small value is typically sufficient. In contrast, problems with strongly non-smooth geometry—such as sharp vertices, or parameters near active-set transitions where the hard-QP derivative changes discontinuously—may benefit from a larger relaxation. Importantly, this approach decouples solution accuracy from gradient smoothing: the forward pass returns a high-accuracy solution of the original QP, while the backward pass returns the derivative of its relaxed surrogate $x_{\kappa_{\text{relax}}}$. In modern automatic differentiation frameworks, these stages can be implemented naturally through customized forward and backward passes [55, 56].

3 Problem Statement

The core computational bottleneck in both solving and differentiating interior-point quadratic programs lies in solving the linear system presented in (5). To reduce the cost of matrix factorization, this full system is typically reduced to a more compact form:

$$\begin{bmatrix} Q & A^\top & G^\top \\ A & 0 & 0 \\ G & 0 & -D(s \oslash z) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = - \begin{bmatrix} r_t \\ r_e \\ r_i - D(z)^{-1} r_{c,\kappa} \end{bmatrix}, \tag{8}$$

where $\oslash$ denotes the elementwise division operator. This compact form exposes a fundamental numerical limitation of the standard primal-dual interior-point formulation: As the solver approaches the optimal solution, complementarity implies $s \odot z \to 0$. Consequently, the entries of the diagonal scaling matrix $D(s \oslash z)$ can become highly disparate. For inactive constraints, $z_i \to 0$ while s_i remains positive, so s_i/z_i diverges. For active constraints, $s_i \to 0$ while z_i remains positive, so s_i/z_i vanishes.

In double-precision arithmetic, this inherent ill-conditioning has been extensively studied and is generally considered manageable [57, 58], as it can be effectively handled using direct matrix factorizations. In single precision, however, this ill-conditioning introduces a severe risk of unstable or inaccurate factorizations, which either lead to poor step directions, unreliable gradients or, even worse, failures resulting from numerical overflows or underflows [41, 42]. Because differentiation reuses the same linear system, any inaccuracy in its factorization propagates directly into the gradients. This numerical vulnerability makes robust single-precision differentiable interior-point solvers difficult, and has led many existing implementations to rely on double precision, preventing them from fully exploiting modern, low-precision machine learning accelerators [40, 28, 27, 51, 29]. To bridge this divide and enable robust differentiable single-precision interior-point solvers, the upcoming section introduces an alternative complementarity representation that effectively circumvents this structural ill-conditioning.

4 Implicit Complementarity

To satisfy the complementarity condition in (3b), we adopt an approach similar to [47, 48, 49], where instead of treating complementarity *explicitly*, we satisfy it *implicitly*, i.e., by construction. We achieve this by introducing an auxiliary variable $v \in \mathbb{R}^p$ and a retraction map

$$b_\kappa : \mathbb{R} \to \mathbb{R}_+ \;\; \text{such that} \;\; b_\kappa(v) \cdot b_\kappa(-v) = \kappa\,, \tag{9}$$

which relates the auxiliary variable by enforcing the following residuals to vanish:

$$r_z \coloneqq z - b_\kappa(v) = 0\,, \tag{10a}$$

$$r_s \coloneqq s - b_\kappa(-v) = 0\,. \tag{10b}$$

4.1 Choosing the retraction map

From all the candidate retraction maps, we pick the *softplus*:

$$b_\kappa(v) = \frac{v + \sqrt{v^2 + 4\kappa}}{2}\,. \tag{11}$$

This specific choice is motivated by the following two properties that its derivative holds

$$\partial_v b_\kappa(v) + \partial_v b_\kappa(-v) = 1\,, \tag{12a}$$

$$0 < \partial_v b_\kappa(v) \leq 1\,. \tag{12b}$$

These will play a fundamental role in the derivations of the underlying linear system that we will use in our approach.

4.2 Newton's Method

Because we operate on the manifold defined by $z \odot s = \kappa$, at each Newton step we target a specific central path parameter value, κ_{target}. This introduces an additional residual:

$$r_\kappa \coloneqq \kappa - \kappa_{\text{target}} = 0\,. \tag{13}$$

To accommodate the new variables, our method solves for the augmented direction $(\Delta x, \Delta y, \Delta z, \Delta s, \Delta v, \Delta\kappa)$. After the derivations in Appendix B.1, the corresponding uncondensed KKT system is:

$$\begin{bmatrix} Q & A^\top & G^\top & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & I & 0 & 0 \\ 0 & 0 & I & 0 & -B_\kappa(v) & -c \\ 0 & 0 & 0 & I & B_\kappa(-v) & -c \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \\ \Delta v \\ \Delta\kappa \end{bmatrix} = - \begin{bmatrix} r_t \\ r_e \\ r_i \\ r_z \\ r_s \\ r_\kappa \end{bmatrix}. \tag{14}$$

where $B_\kappa(v) = D(\partial_v b_\kappa(v))$ and $c = \partial_\kappa b_\kappa(v)$. By exploiting the aforementioned properties of the retraction map in (12), we can reduce it into the following form[1]:

$$\begin{bmatrix} Q - G^\top G & G^\top & A^\top \\ G & -B_\kappa(-v) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta v \\ \Delta y \end{bmatrix} = - \begin{bmatrix} r_t - G^\top(r_i + r_z - r_s) \\ r_i - r_s - c\, r_\kappa \\ r_e \end{bmatrix}. \tag{15}$$

The matrix of this reduced system exhibits two crucial properties: (i) it is symmetric, allowing for efficient $LDL^\top$ factorization [59], and (ii) all of its block components remain constant across iterations, with the sole exception of $B_\kappa(-v)$. Notably, this middle block is diagonal and strictly bounded between 0 and 1 due to the derivative range of the retraction map (12b). These properties yield two primary advantages: First, the combination of symmetry and a predominantly static structure—where the only varying component is bounded and diagonal—enables a wide array of efficient linear solver strategies [48]: although the changing block $B_\kappa(-v)$ requires a new numerical factorization at every iteration, the fixed sparsity pattern allows the symbolic analysis and orderings to be computed only once. Second, the conditioning of the system no longer degrades along the central path. Denoting the matrix in (15) by K_κ, three observations hold: (a) since the entries of $B_\kappa(-v)$ lie in $(0, 1)$, $\|K_\kappa\|$ is bounded by the norm of its fixed part plus one for every $\kappa > 0$, whereas the block $D(s \oslash z)$ in (8) grows without bound as $\kappa \to 0$; (b) as $\kappa \to 0$, the entries of $B_\kappa(-v)$ tend to 0 on active and to 1 on inactive inequality constraints, so K_κ converges to the KKT matrix of the active-set system; and (c) under linear independence and second-order sufficiency, this limit is nonsingular, so $\operatorname{cond}(K_\kappa)$ is bounded uniformly in κ. We stress that this is a limiting argument rather than a quantitative bound: it removes the complementarity-induced ill-conditioning, while ill-conditioning inherited from Q, G, and A is unaffected (see Appendix E.4 for empirical conditioning measurements). Consequently, the linear system in (15) is structurally robust against the numerical instabilities that hinder standard interior-point methods in low-precision arithmetic.

[1] For alternative partially condensed forms see Appendix B.3.

5 Differentiable Implicit Primal-Dual Interior-Point Methods

With a low-precision-compliant linear system established, we return to the differentiation procedure introduced in Section 2.3. As previously discussed, this three-step approach allows us to compute high-quality solutions to tight tolerances while evaluating smooth gradients at a relaxed point.

Here, we adapt this procedure to the implicit complementarity case and show how this sequence can be implemented as custom forward and backward passes compliant with modern automatic differentiation frameworks [55, 56].

5.1 Solving the QP to high accuracy

To solve the quadratic program to a tight tolerance, we integrate the implicit complementarity representation from the previous section into a primal-dual interior-point pipeline. An algorithmic overview is provided in Algorithm 1. Two key aspects are worth highlighting: First, low-precision KKT solves must avoid the uncondensed form, relying exclusively on the partially condensed system in (15) (see Appendix B.2). Second, the target parameter κ_{target} is updated via a constant reduction factor σ. However, as in standard IPMs, this naturally can be extended to adaptive strategies, such as predictor-corrector schemes [61] (see Appendix B.4).

5.2 Relaxing the QP solution

The relaxation procedure is detailed in Algorithm 2. Its underlying mechanism closely mirrors that of Algorithm 1, with two primary distinctions: First, initialization is no longer required, as we warm-start the procedure directly from the optimal solution (x^*, y^*, z^*, s^*) obtained via Algorithm 1. Second, rather than driving the central path parameter to zero ($\kappa \to 0$), we increase it toward a target relaxation value ($\kappa \to \kappa_{\text{relax}}$). Because this relaxation essentially traces the central path in reverse, the symbolic analysis and ordering of $\tilde{K}$ from the forward solve in Algorithm 1 carry over unchanged, and the final factorization at κ_{relax} is the one reused by the backward pass.

5.3 Computing Gradients of the QP

We compute the gradients at the relaxed point in a similar fashion to [27, 29]. Instead of naively constructing the full Jacobian matrices—which can easily become prohibitively large—we apply the

Algorithm 1 Solving a Quadratic Program with Implicit Complementarity

function SOLVE_QP($Q, q, A, b, G, h; \sigma, \text{tol}, \text{max_iter}$)
  $(x, s, z, y) \leftarrow$ INITIALIZE(Q, q, A, b, G, h) ▷ same as in [60, 53]
  for $i \leftarrow 1 : \text{max_iter}$ **do**

    /* manifold coordinates */
    $v \leftarrow z - s$
    $\kappa \leftarrow \left(s^\top z\right)/p, \qquad \kappa_{\text{target}} \leftarrow \sigma\kappa$

    /* residuals and check convergence */
    $r \leftarrow (r_t, r_e, r_i, r_z, r_s, r_\kappa)$ ▷ eqs. (3), (10) and (13)
    if $\|(r_t, r_c, r_i, r_e)\|_\infty < \text{tol}$ **then return** x, y, z, s ▷ KKT residuals, eq. (3)

    /* newton step */
    $\tilde{K} \leftarrow$ PRECOMPUTE_KKT_FACTORS(Q, A, G, v, κ) ▷ symbolic analysis is cacheable
    $(\Delta x, \Delta y, \Delta z, \Delta s, \Delta v, \Delta\kappa) \leftarrow$ SOLVE_KKT$(\tilde{K}, r)$ ▷ eq. (14)

    /* step on (x, y, v, κ), then retract */
    $\alpha \leftarrow$ STEP_SIZE$(s, z, \Delta s, \Delta z)$ ▷ eq. (6)
    $x \leftarrow x + \alpha\Delta x, \qquad y \leftarrow y + \alpha\Delta y$
    $v \leftarrow v + \alpha\Delta v, \qquad \kappa \leftarrow \kappa + \alpha\Delta\kappa$
    $z, s \leftarrow$ RETRACTION_MAP(v, κ) ▷ eqs. (10) and (11)
  end for
end function

Algorithm 2 Relaxing an Implicit Quadratic Program to κ_{relax}

function RELAX_QP($Q, q, A, b, G, h, x, y, z, s, \kappa_{\text{relax}}; \tilde{K}, \text{tol}, \text{max_iter}$)
  for $i \leftarrow 1 : \text{max_iter}$ **do**

    /* manifold coordinates */
    $v \leftarrow z - s$
    $\kappa \leftarrow \left(s^\top z\right)/p, \qquad \kappa_{\text{target}} \leftarrow \kappa_{\text{relax}}$

    /* residuals and check convergence */
    $r \leftarrow (r_t, r_e, r_i, r_z, r_s, r_\kappa)$ ▷ eqs. (3), (10) and (13)
    if $\|r\|_\infty < \text{tol}$ **then return** $x, y, z, s, \tilde{K}$ ▷ includes $r_\kappa = \kappa - \kappa_{\text{relax}}$

    /* newton step */
    $\tilde{K} \leftarrow$ PRECOMPUTE_KKT_FACTORS(Q, A, G, v, κ) ▷ symbolic analysis reused from Alg. 1
    $(\Delta x, \Delta y, \Delta z, \Delta s, \Delta v, \Delta\kappa) \leftarrow$ SOLVE_KKT$(\tilde{K}, r)$ ▷ eq. (14)

    /* step on (x, y, v, κ), then retract */
    $\alpha \leftarrow$ STEP_SIZE$(s, z, \Delta s, \Delta z)$ ▷ eq. (6)
    $x \leftarrow x + \alpha\Delta x, \qquad y \leftarrow y + \alpha\Delta y$
    $v \leftarrow v + \alpha\Delta v, \qquad \kappa \leftarrow \kappa + \alpha\Delta\kappa$
    $z, s \leftarrow$ RETRACTION_MAP(v, κ) ▷ eqs. (10) and (11)
  end for
end function

implicit function theorem to efficiently compute only the specific vector-Jacobian products required by the backward pass. The resulting procedure is detailed in Algorithm 3.

Two key aspects of this procedure are worth noting: First, because the gradients are evaluated at κ_{relax}, we can directly reuse the final matrix factorization computed in Algorithm 2. Second, the implicit reparameterization naturally absorbs the corresponding Jacobian factors, eliminating the need for the explicit multiplier scaling and unscaling steps required in standard primal-dual methods.

6 Numerical Experiments

We evaluate our approach across four distinct benchmarks, structuring this section accordingly. In each experiment, we compare the conventional primal-dual IPM baseline—where complementarity is handled *explicitly* (visualized in blue)—against our *implicit* formulation (visualized in orange). For clarity, we denote single-precision and double-precision arithmetic as f32 and f64, respectively.

To ensure a controlled and equitable comparison, we use the JAX-based framework of [7] as a baseline for state-of-the-art differentiable IPMs [27, 51, 62]. We extend this codebase with our implicit method, so that the two paradigms are evaluated within the same architecture: both share

Algorithm 3 Computing Gradients Through an Implicit Relaxed QP

function COMPUTE_QP_GRADIENTS$(Q, q, A, b, G, h, x, y, z, s, \kappa_{\text{relax}}, \nabla_x\ell; \tilde{K})$
  /* compute differentials */
  $r \leftarrow (-\nabla_x\ell, 0, 0, 0, 0, 0)$ ▷ $\nabla_x\ell$: incoming adjoint of the downstream scalar loss ℓ
  $(dx, dy, dz, ds, _, _) \leftarrow$ SOLVE_KKT$(\tilde{K}, r)$ ▷ $\tilde{K}$ is the final factorization from Alg. 2

  /* parameter gradients */
  $\nabla_Q\ell \leftarrow \dfrac{(dx)x^\top + x(dx)^\top}{2}$
  $\nabla_q\ell \leftarrow dx$
  $\nabla_A\ell \leftarrow (dy)x^\top + y(dx)^\top$
  $\nabla_b\ell \leftarrow -(dy)$
  $\nabla_G\ell \leftarrow (dz)x^\top + z(dx)^\top$
  $\nabla_h\ell \leftarrow -dz$
  return $\nabla_Q\ell, \nabla_q\ell, \nabla_A\ell, \nabla_b\ell, \nabla_G\ell, \nabla_h\ell$
end function

the same initialization, stopping criteria, precision, and differentiation pipeline, differing only in the representation of complementarity. Section 6.1 uses true single-precision arithmetic on an NVIDIA RTX 3080; Sections 6.2–6.4 use the TF32 default of current GPUs on an RTX 5090 (see Appendix D).

6.1 Solver Benchmark

Before turning to differentiation, we position the forward solve against external solvers on the five problem classes of the OSQP benchmark suite [63]—random QPs, portfolio, lasso, SVM, and optimal control—with five sizes and 20 random instances per size, for a total of 500 QPs (dimensions and timings are given in Appendix E). Besides the controlled explicit baseline, we include cuClarabel [51], a GPU interior-point solver, and BoxOSQP [64], a GPU first-order (ADMM) solver. Every returned solution is re-verified in f64 against the original problem data, and a solve counts as successful only if the primal and dual residuals, the relative duality gap, and the complementarity violation all meet the requested tolerance. In f64, all four solvers succeed on 500/500 instances at every tolerance. In f32 (Table 2 in Appendix E), implicit succeeds on 500/500 instances at 10^{-3} and 10^{-5}, and as the tolerance tightens the gap widens: at 10^{-6} it succeeds on 497 instances against 154 for explicit f32 and 413 for the best external solver, with a median relative objective error of 3.1×10^{-8}. This reliability has a cost: implicit f32 is 2.2–5.4$\times$ slower than explicit f32 in median solve time, a tradeoff we return to in Section 7. Appendix E.3 repeats the benchmark under TF32.

6.2 Gradient Fidelity and Robustness in Single-Precision

In this experiment, we investigate the gradient quality and robustness of the implicit and explicit paradigms in single-precision (f32) arithmetic. Using the Euclidean projection onto a polytope against an exact active-set reference, we assess whether the projection map's derivative remains accurate and numerically stable as the solver relaxation parameter κ_{relax} varies, the active set grows, and problem dimensionality increases[2].

Results are depicted in Fig. 1. Sweeping over problem dimensions n and constraints m (first column), the implicit method's gradient error stays at the theoretical minimum dictated by the relaxation floor ($\kappa_{\text{relax}} = 10^{-4}$). Conversely, the explicit method's accuracy degrades significantly with scale: for $n \geq 10$, it frequently triggers multiple active sets, causing severe numerical instability where about 60% of computations return NaNs (see Appendix F.4 for a detailed trace). The implicit method, in contrast, maintains a 0% failure rate and demonstrates total robustness to problem scale.

Evaluating the impact of the relaxation parameter κ_{relax} at $n = 20$ (second column) reveals that the explicit method's error stagnates near 10^{-1}, while the implicit method faithfully tracks the κ_{relax}-

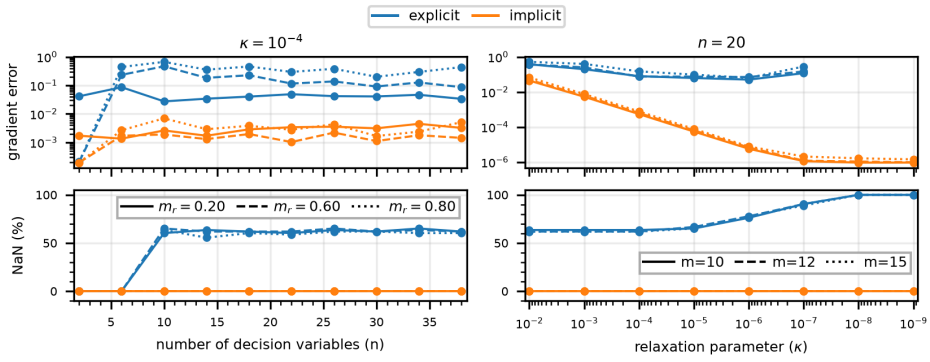

Figure 1: Implicit complementarity enables accurate and robust single-precision gradients. *Left*: As problem dimension and active-set size increase, the implicit method maintains low gradient error and remains fully robust, whereas the explicit formulation fails about 60% of the time for $n \geq 10$. Here, m_r denotes the ratio of active inequality constraints m to decision variables n. *Right*: As κ_{relax} decreases, the implicit method continues to track the relaxation-limited gradient accuracy, whereas the explicit f32 formulation plateaus in accuracy and exhibits an escalating failure rate. Reported metrics are medians over three random seeds.

[2]See Appendix F for further details on the problem formulation and experiment design.

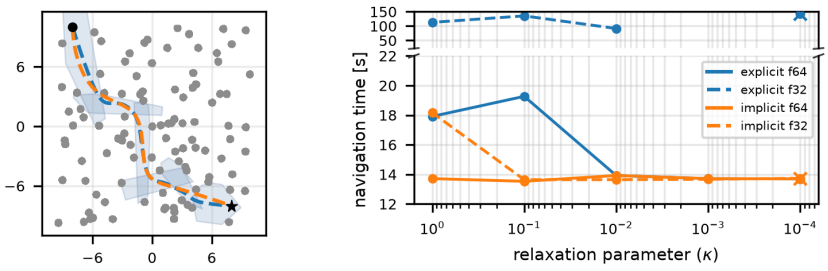

Figure 2: Bilevel trajectory optimization. Implicit f32 matches f64 at every relaxation except the loosest, whereas explicit f32 is unusable. *Left:* Trajectories in forest environment. *Right:* Navigation time for different relaxations. Crosses refer to the trajectories plotted in the left.

allowed minimum until plateauing at 10^{-6}. Moreover, as relaxation tightens, the explicit method becomes increasingly prone to failure, with NaN detection scaling from about 60% to 100%. These findings demonstrate the implicit approach's superior accuracy and reliability in single-precision settings, particularly when high-dimensional constraints or tight relaxations are required.

6.3 Bilevel Optimization

Following our synthetic experiments, we evaluate our approach on a real-world bilevel trajectory optimization problem from robotics. The inner QP smooths the trajectory while enforcing kinematic and collision constraints, whereas the outer loop optimizes navigation time [65, 66, 67]. Modeling the trajectory as a composite Bezier curve allows geometric safety to be enforced via linear constraints on control points, yielding a QP with coefficients that depend smoothly on segment durations (see Appendix G). For a fixed sequence of durations $T = (T_1, \ldots, T_K)$, the inner QP minimizes a weighted smoothness objective subject to safety, boundary, and derivative constraints. The outer level then updates T to minimize the inner objective alongside a linear time penalty. Because this outer objective depends directly on the inner QP's solution, optimizing T requires differentiating through the solver.

We optimize the outer loop via L-BFGS [68, 33], comparing the usefulness of gradients provided by the explicit and implicit paradigms. We sweep across multiple relaxation parameters in single and double precision for two environments: a forest and an office. Since both scenarios yield similar behavior, we present the forest environment here (Fig. 2) and defer the office to the appendix (Fig. 5). These sweeps reveal two key insights: First, at loose relaxations some runs stop early with a worse navigation time, whereas tightening the parameter beyond 10^{-2} offers no further gain. Second, although both methods perform equivalently in double precision, the explicit method in single precision produces no usable result at any relaxation: its navigation time stays an order of magnitude above the rest, and at $\kappa_{\text{relax}} = 10^{-3}$ its first gradient is not finite. In contrast, the implicit method in single precision remains robust, performing on par with its double-precision counterpart. As in our synthetic study, the implicit method thus delivers reliable gradients in both precisions.

6.4 Learning a Multi-Agent Safety Filter

While the bilevel trajectory optimization serves as a compelling application, one might argue it represents a specialized use case. In standard deep learning settings, stochastic optimizers such as SGD [69] and Adam [70] are often robust to noisy gradients, reducing the need for highly accurate derivatives [71]. This raises a natural question: are explicit single-precision (f32) gradients truly insufficient, and does the improved gradient quality and robustness of our implicit approach offer a tangible advantage in end-to-end learning pipelines?

To investigate this question, we consider a centralized multi-agent safety-filter learning task. Here, a neural network predicts a nominal control law alongside state-dependent control barrier function (CBF) gains, and a differentiable quadratic programming (QP) layer projects this proposal onto a safe set (see Appendix H for formulation and implementation details) [72]. This environment is an ideal testbed because it directly addresses the aforementioned questions while allowing us to evaluate the

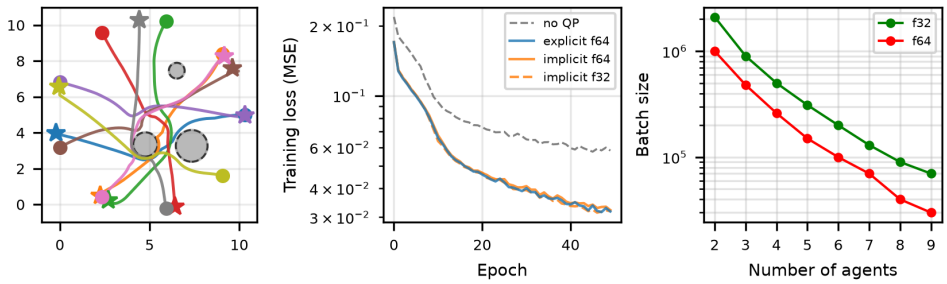

Figure 3: End-to-end learning of a multi-agent safety filter. Implicit f32 matches f64 training stability (unlike explicit f32, which fails entirely) and allows for double the maximum batch size.

high-dimensional regime explored in our synthetic case study; specifically, increasing the number of agents scales the QP's decision variables linearly and its inequality constraints quadratically.

We evaluate the learning process scaling from 2 to 9 agents in an environment with 3 obstacles. The results are summarized in Fig. 3. The first column illustrates an exemplary scenario with 9 agents, while the second displays the corresponding training loss curves. As a baseline, we also include a vanilla MLP lacking a QP safety filter. Notably, the implicit f32, implicit f64, and explicit f64 formulations exhibit near-identical, stable performance. Crucially, however, the explicit f32 method is omitted from these plots because it fails (NaNs) in the first training batch at every agent count. This contrast demonstrates how our implicit approach successfully unlocks the use of single-precision arithmetic in end-to-end learning with differentiable QPs under the default arithmetic of modern GPUs, where the standard formulation fails.

This capability offers a significant practical advantage, as modern machine learning hardware is natively optimized for single-precision operations. As illustrated in the third column of Fig. 3, the implicit f32 method accommodates GPU batch sizes exactly twice as large as its f64 counterpart, definitively demonstrating superior memory efficiency and scalability.

7 Limitations

The reliability of the implicit formulation comes at a computational cost: the reduced system (15) contains $Q - G^\top G$, which introduces fill-in when Q and G are sparse, and retains the p-dimensional variable v, so it can be larger and costlier to factorize than the fully condensed system of a conventional IPM; our prototype is 2.2–5.4$\times$ slower than explicit f32 in median solve time (Section 6.1), a gap that sparse or tailored $LDL^\top$ factorizations can reduce but not remove. The conditioning argument of Section 4 is a limiting one: it removes the complementarity-induced blow-up but provides no quantitative bound, and ill-conditioning inherited from Q, G, and A is unaffected. The method assumes a feasible convex QP and performs no infeasibility detection, and we claim no global convergence or iteration-complexity guarantees beyond the local equivalence with the perturbed KKT conditions for fixed κ. Finally, the three-stage architecture extends conceptually to other symmetric cones through a cone-specific retraction (Appendix I), but an efficient, well-conditioned realization beyond QPs is not established here.

8 Conclusions

In this paper, we introduced a differentiable interior-point method that resolves the numerical instabilities of solving and differentiating convex quadratic programs in single-precision arithmetic. By replacing standard explicit complementarity with an implicit representation, our approach avoids the severe ill-conditioning that makes the standard formulation unreliable in single precision. Extensive evaluations—spanning synthetic benchmarks, bilevel trajectory optimization, and multi-agent safety filter learning—demonstrate that our implicit method matches the gradient accuracy and robustness of double-precision solvers. Crucially, by unlocking reliable low-precision operations, our formulation fully leverages the memory and throughput advantages of modern hardware, establishing a scalable foundation for embedding constrained optimization into machine learning pipelines.

References

[1] Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2589–2594. IEEE, 2014.

[2] Lars Blackmore. Autonomous precision landing of space rockets. In *Frontiers of engineering: reports on leading-edge engineering from the 2016 symposium*, volume 46, pages 15–20. The Bridge Washington, DC, USA, 2016.

[3] Patrick Varin and Scott Kuindersma. A constrained kalman filter for rigid body systems with frictional contact. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 474–490. Springer, 2018.

[4] 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 Magazine*, 42(5):40–113, 2022.

[5] Fausto Vega, Jon Arrizabalaga, Ryan Watson, and Zachary Manchester. Convex maneuver planning for spacecraft collision avoidance, 2025.

[6] Mihai Anitescu. Optimization-based simulation of nonsmooth rigid multibody dynamics. *Mathematical Programming*, 105(1):113–143, 2006.

[7] Kevin Tracy, Taylor A Howell, and Zachary Manchester. Differentiable collision detection for a set of convex primitives. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3663–3670. IEEE, 2023.

[8] Tao Pang and Russ Tedrake. A convex quasistatic time-stepping scheme for rigid multibody systems with contact and friction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6614–6620. IEEE, 2021.

[9] Martin Beckmann, C. B. McGuire, and Christopher B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, 1956.

[10] Herbert Scarf. A min-max solution of an inventory problem. *Studies in the Mathematical Theory of Inventory and Production*, pages 201–209, 1958.

[11] Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of operations research*, 23(4):769–805, 1998.

[12] Mark Rubinstein. Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance*, 57(3):1041–1045, 2002.

[13] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[15] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.

[16] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[17] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[18] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[19] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Priya L. Donti, Brandon Amos, and J. Zico Kolter. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, 2017.

[21] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[22] Tim Salzmann, Jon Arrizabalaga, Joel Andersson, Marco Pavone, and Markus Ryll. Learning for CasADi: Data-driven models in numerical optimization. In Alessandro Abate, Mark Cannon, Kostas Margellos, and Antonis Papachristodoulou, editors, *Proceedings of the 6th Annual Learning for Dynamics; Control Conference*, volume 242 of *Proceedings of Machine Learning Research*, pages 541–553. PMLR, 15–17 Jul 2024.

[23] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, 2023.

[24] Angel Romero, Elie Aljalbout, Yunlong Song, and Davide Scaramuzza. Actor–critic model predictive control: Differentiable optimization meets reinforcement learning for agile flight. *IEEE Transactions on Robotics*, 42:673–692, 2026.

[25] Emre Adabag, Marcus Greiff, John Subosits, and Thomas Lew. Differentiable model predictive control on the gpu, 2025.

[26] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019.

[27] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.

[28] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[29] Kevin Tracy and Zachary Manchester. On the differentiability of the primal-dual interior-point method. *arXiv preprint arXiv:2406.11749*, 2024.

[30] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[31] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[32] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.

[33] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2 edition, 2006.

[34] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1–2):281–302, 2000.

[35] Stephen J. Wright. Effects of finite-precision arithmetic on interior-point methods for nonlinear programming. *SIAM Journal on Optimization*, 12(1):36–78, 2001.

[36] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[37] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018.

[38] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[39] Massimiliano Fasi, Nicholas J. Higham, Mantas Mikaitis, and Srikara Pranesh. Numerical behavior of nvidia tensor cores. *PeerJ Computer Science*, 7:e330, 2021.

[40] Azzam Haidar, Harun Bayraktar, Stanimire Tomov, Jack Dongarra, and Nicholas J Higham. Mixed-precision iterative refinement using tensor cores on gpus to accelerate solution of linear systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2243), 2020.

[41] Ahmad Abdelfattah, Hartwig Anzt, Erik G Boman, Erin Carson, Terry Cojean, Jack Dongarra, Alyson Fox, Mark Gates, Nicholas J Higham, Xiaoye S Li, et al. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications*, 35(4):344–369, 2021.

[42] Nicholas J Higham and Theo Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

[43] Erin Carson and Nicholas J Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017.

[44] Gianluca Frison and Moritz Diehl. Hpipm: a high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine*, 53(2):6563–6569, 2020.

[45] Tecla De Luca, Francisco Facchinei, and Christian Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. *Mathematical Programming*, 75(3):407–439, 1996.

[46] Dominic Liao-McPherson and Ilya Kolmanovsky. Fbstab: A proximally stabilized semismooth algorithm for convex quadratic programming. *Automatica*, 113:108801, 2020.

[47] Frank Permenter. Log-domain interior-point methods for convex quadratic programming. *Optimization Letters*, 17(7):1613–1631, 2023.

[48] Jon Arrizabalaga and Zachary Manchester. Implicit primal-dual interior-point methods for quadratic programming. *arXiv preprint arXiv:2604.00364*, 2026.

[49] Arun L Bishop, Micah I Reich, and Zachary Manchester. Complementarity by construction: A lie-group approach to solving quadratic programs with linear complementarity constraints. *arXiv preprint arXiv:2604.11991*, 2026.

[50] Andrew Butler and Roy H Kwon. Efficient differentiable quadratic programming layers: an admm approach. *Computational Optimization and Applications*, 84(2):449–476, 2023.

[51] Yuwen Chen, Danny Tse, Parth Nobel, Paul Goulart, and Stephen Boyd. Cuclarabel: Gpu acceleration for a conic optimization solver. *arXiv preprint arXiv:2412.19027*, 2024.

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

[53] Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[54] Steven G. Krantz and Harold R. Parks. *The Implicit Function Theorem: History, Theory, and Applications*. Birkhäuser, 2012.

[55] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[57] Margaret H Wright. Ill-conditioning and computational error in interior methods for nonlinear programming. *SIAM Journal on Optimization*, 9(1):84–111, 1998.

[58] Michele Benzi, Gene H Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14:1–137, 2005.

[59] Gene H Golub and Charles F Van Loan. Matrix computations. johns hopkins studies in the mathematical sciences, 1996.

[60] Lieven Vandenberghe. The cvxopt linear and quadratic cone program solvers. *Online: http://cvxopt. org/documentation/coneprog. pdf*, 53, 2010.

[61] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[62] Shane Barratt, Parth Nobel, and Steven Diamond. Moreau: GPU-native differentiable optimization, 2026.

[63] 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.

[64] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[65] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. Ieee, 2011.

[66] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.

[67] Jesus Tordesillas, Brett T Lopez, Michael Everett, and Jonathan P How. Faster: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Transactions on Robotics*, 38(2):922–938, 2021.

[68] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[69] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[70] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[71] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

[72] Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. Barriernet: Differentiable control barrier functions for learning of safe robot control. *IEEE Transactions on Robotics*, 39(3):2289–2307, 2023.

A Implicit Function Theorem in Primal-Dual Interior-Point QPs

Consider variables $w \in \mathbb{R}^a$ and parameters $\theta \in \mathbb{R}^b$, and suppose that an equilibrium point w^* is defined implicitly by

$$r(w^*, \theta) = 0. \tag{16}$$

Linearizing the residual r around w^* gives

$$\frac{\partial r}{\partial w}\delta w + \frac{\partial r}{\partial \theta}\delta\theta = 0. \tag{17}$$

Solving this equation for the parameter derivatives yields:

$$\frac{\partial w}{\partial \theta} = -\left(\frac{\partial r}{\partial w}\right)^{-1}\frac{\partial r}{\partial \theta}. \tag{18}$$

Applying this to the present formulation, the KKT conditions in (3) are treated as the residual r, the primal-dual variables (x, s, z, y) as w, and the problem data as the parameters θ. The implicit function theorem can therefore be used to differentiate the optimizer with respect to the problem data. Importantly, the linear system associated with (18) has the same structure as the Newton system in (5). As a result, these derivatives can be computed efficiently by reusing the same linear-algebra framework, and potentially the same matrix factorizations, already available inside the interior-point solver.

Applying the theorem requires $\partial r/\partial w$ to be nonsingular at the point of differentiation. For a convex QP this holds when A has full row rank and $Q + G^\top D(z \oslash s)G$ is positive definite on the null space of A; when these conditions fail, unique classical sensitivities require regularization or a selection rule.

B The KKT system of a QP with implicit complementarity

B.1 Derivation of the uncondensed form

We start by linearizing the implicit complementarity residuals from (10) with respect to v and κ:

$$r_z^+ \approx r_z + \Delta z - B_\kappa(v)\,\Delta v - c\,\Delta\kappa\,, \tag{19a}$$

$$r_s^+ \approx r_s + \Delta s + B_\kappa(-v)\Delta v - c\Delta\kappa\,, \tag{19b}$$

$$r_\kappa^+ \approx r_\kappa + \Delta\kappa\,, \tag{19c}$$

where $B_\kappa(v) = D(\partial_v b_\kappa(v))$ and $c = \partial_\kappa b_\kappa(v)$. Combining these residuals with the ones in (3) yields the following augmented linear system:

$$\begin{bmatrix} Q & A^\top & G^\top & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & I & 0 & 0 \\ 0 & 0 & I & 0 & -B_\kappa(v) & -c \\ 0 & 0 & 0 & I & B_\kappa(-v) & -c \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \\ \Delta v \\ \Delta\kappa \end{bmatrix} = - \begin{bmatrix} r_t \\ r_e \\ r_i \\ r_z \\ r_s \\ r_\kappa \end{bmatrix}. \tag{20}$$

B.2 Numerical issues of the uncondensed form in low-precision arithmetic

The uncondensed system in eq. (20) is problematic in low-precision arithmetic. This is because the auxiliary increment Δv influences the Newton step primarily through its coupling with Δz via $B_\kappa(v)$. Because Δz enters the stationarity equation directly, this indirect dependency becomes a critical bottleneck. In particular, as κ vanishes and a constraint becomes inactive, the diagonal entries of $B_\kappa(v)$ approach zero. Under low-precision regimes, this causes the signal from Δv to be effectively lost, leading to a breakdown in the coupling required for accurate stationarity updates.

B.3 Partially condensed forms

The uncondensed system can be reduced into two distinct partially condensed forms. The first and most intuitive reduction follows the standard IPM structure:

$$\begin{bmatrix} Q & G^\top B_\kappa(v) & A^\top \\ G & -B_\kappa(-v) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta v \\ \Delta y \end{bmatrix} = - \begin{bmatrix} r_t - G^\top (r_z + c\, r_\kappa) \\ r_i - r_s - c\, r_\kappa \\ r_e \end{bmatrix}. \tag{21}$$

While this system closely resembles the traditional formulation in eq. (8), it remains explicitly dependent on $B_\kappa(v)$. Consequently, it inherits the numerical instabilities detailed in Appendix B.2, making it unreliable for low-precision computations.

To address these limitations, we propose the second candidate introduced in eq. (15):

$$\begin{bmatrix} Q - G^\top G & G^\top & A^\top \\ G & -B_\kappa(-v) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta v \\ \Delta y \end{bmatrix} = - \begin{bmatrix} r_t - G^\top (r_i + r_z - r_s) \\ r_i - r_s - c\, r_\kappa \\ r_e \end{bmatrix}. \tag{22}$$

Although this approach densifies the Hessian, it offers the significant advantages discussed in Section 4: the system is symmetric, predominantly static, and free of complementarity-induced ill-conditioning. Most importantly, by eliminating the reliance on $B_\kappa(v)$, it bypasses the numerical issues of the uncondensed form, establishing itself as the most robust candidate for low-precision arithmetic.

B.4 Convergence and adaptive barrier updates

For every $\kappa > 0$, b_κ is smooth, so Algorithm 1 takes ordinary Newton steps within a primal-dual path-following scheme; it is not a semismooth Newton method. For fixed κ, the parameterization $z = b_\kappa(v)$, $s = b_\kappa(-v)$ is equivalent to the conventional perturbed condition $z \odot s = \kappa \mathbf{1}$, so the Newton iterations target the same central-path point and retain local quadratic convergence under the usual nondegeneracy assumptions. The change of coordinates nevertheless alters the iterates and the globalization, and we claim neither identical global convergence behavior nor the polynomial iteration-complexity guarantees of classical short-step IPMs.

Adaptive barrier updates such as Mehrotra's predictor-corrector scheme [61] require no modification of the linear system: the corrector reuses the same matrix $\tilde{K}$ with a second right-hand side, and since $B_\kappa(-v)$ depends only on the current iterate and κ, the centering and corrector heuristics change the right-hand side but not the spectrum of the factorized system, so the argument of Section 4 is unaffected. We implemented a Mehrotra-like predictor-corrector and observed that an aggressive affine predictor targeting $\kappa \to 0$ degrades the corrector step, because z and s are recovered through the retraction rather than updated directly; with a less aggressive predictor the scheme contributed little, so we kept the constant-reduction update of Algorithm 1.

C Implementation of the derivative of the softplus retraction map

To avoid catastrophic cancellation (when $v < 0$ and $v^2 >> \kappa$) in the retraction map $b_\kappa(v)$ and its derivative $\partial_v b_\kappa(v)$, the software implementation of the softplus retraction map and its derivative needs to be carefully conducted:

$$b_\kappa(v) = \begin{cases} \frac{v + \sqrt{v^2 + 4\kappa}}{2} & \text{if } v \geq 0 \\ \frac{2\kappa}{\sqrt{v^2 + 4\kappa} - v} & \text{if } v < 0 \end{cases} \tag{23a}$$

$$\partial_v b_\kappa(v) = \begin{cases} \frac{1}{2}\left(1 + \frac{v}{\sqrt{v^2 + 4\kappa}}\right) & \text{if } v \geq 0 \\ \frac{2\kappa}{v^2 + 4\kappa - v\sqrt{v^2 + 4\kappa}} & \text{if } v < 0 \end{cases} \tag{23b}$$

D Experimental Setup

Both solvers run inside the same codebase and share one pipeline: the same initialization, the same stopping test, which requires the KKT residual and the relative duality gap to fall below the requested

tolerance, and, within each experiment, the same tolerance, iteration cap, and relaxation. The explicit baseline additionally carries the safeguards it needs in order to terminate rather than stall at the iteration cap: its stopping test is evaluated on the true iterate, its slack and dual variables are floored at 300 machine epsilons inside the Newton step, its Schur complement is regularized, and during its relaxation stage the slack and dual variables are floored at $\sqrt{\kappa_{\mathrm{relax}}}$. The implicit solver has no counterpart to these safeguards.

On NVIDIA GPUs, JAX executes f32 matrix products in TF32 by default, rounding the inputs to a 10-bit mantissa. Both solvers read one global setting that selects the arithmetic of every matrix product inside them, true single precision or TF32, and every result file records the mode it was produced in; f64 runs are unaffected. The solver benchmark of Section 6.1 uses true single precision, for consistency with the external solvers, whose matrix arithmetic we do not control. Sections 6.2–6.4 use TF32, the arithmetic a practitioner obtains by default. Under either mode, products whose operands are both constant problem data, namely $G^\top G$ and $G^\top h$, are formed once per solve at full precision in both solvers, and every product that involves an iterate follows the mode.

Section 6.1 was run on an NVIDIA GeForce RTX 3080 (12 GB). Unless stated otherwise, Sections 6.2–6.4 were run on an NVIDIA GeForce RTX 5090 (32 GB) with JAX 0.12.0.1. Timings are therefore not comparable across the two groups of experiments.

E Solver Benchmark Details

E.1 Protocol

We use the five problem classes of the OSQP benchmark suite [63]: random QPs, portfolio optimization, lasso, support vector machine (SVM) classification, and optimal control. Each class is generated at five sizes with 20 random instances per size, giving 100 QPs per class and 500 in total; the resulting problem dimensions are listed in Table 1. Every instance is solved at three tolerances, 10^{-3}, 10^{-5}, and 10^{-6}, in f64 and in f32, here with true single-precision matrix products (Appendix D).

Besides the controlled explicit baseline, we include two GPU solvers through their public implementations: cuClarabel [51], an interior-point solver, and BoxOSQP from JAXopt [64], a first-order (ADMM) solver. All returned solutions are re-evaluated in f64 using the original problem data. A solve is counted as successful only if the maximum of the primal-feasibility residual, the dual-feasibility residual, the relative duality gap, and the complementarity residual is below the requested tolerance. Solve times are medians over the instances that reached the requested tolerance and exclude compilation.

Table 1: Problem classes of the OSQP benchmark suite used in Section 6.1. Each class has five sizes with 20 random instances per size; n, m, and p denote the number of variables, equality constraints, and inequality constraints.

Class	Sizes	n	m	p
random QP	10, 15, 22, 33, 50	10–50	0	100–500
portfolio	5, 7, 9, 12, 15	505–1515	6–16	1000–3000
lasso	5, 7, 9, 12, 15	510–1530	500–1500	10–30
SVM	10, 13, 17, 23, 30	1010–3030	0	2000–6000
control	10, 13, 17, 23, 30	160–480	110–330	320–960

E.2 Results

In f64, all four solvers succeed on 500/500 instances at every tolerance. The f32 success counts are reported in Table 2. At 10^{-6}, implicit f32 succeeds on 497/500 instances against 154/500 for explicit f32; the three unsuccessful implicit runs remain finite, with a worst externally evaluated KKT residual of 2.83×10^{-6} and no NaNs. Across all successful implicit f32 solutions, the median relative objective error is 3.1×10^{-8}, the median constraint violation 6.0×10^{-8}, and the median dual residual 1.2×10^{-7}. The largest differences between the two formulations arise on portfolio, SVM, and control, where explicit f32 solves 0/100 instances at 10^{-6} in each class, compared with 99/100, 100/100, and 98/100 for implicit f32.

Table 2: Successful f32 solves out of 500 OSQP-benchmark QPs, re-verified in f64. In f64 all four solvers succeed on 500/500 at every tolerance.

tol	implicit (ours)	explicit	cuClarabel	BoxOSQP
10^{-3}	**500** (100%)	496 (99.2%)	498 (99.6%)	460 (92.0%)
10^{-5}	**500** (100%)	272 (54.4%)	486 (97.2%)	460 (92.0%)
10^{-6}	**497** (99.4%)	154 (30.8%)	413 (82.6%)	384 (76.8%)

Table 3 reports the median f32 solve times. The runtime and success results must be interpreted jointly: explicit f32 has the lowest median runtime on four of the five classes but fails on most instances at 10^{-6}, where its runs terminate at the iteration cap rather than at a solution. cuClarabel is faster than our current implicit implementation on several classes while attaining a lower f32 success rate at the tightest tolerance, and BoxOSQP is substantially slower, running into 20,000 or more iterations per solve. These comparisons reflect both algorithmic and implementation differences; the controlled explicit-versus-implicit comparison isolates the effect of the complementarity representation. The benchmark scripts and per-instance records of the implicit and explicit solvers are provided in the supplementary material; cuClarabel and BoxOSQP were run through their public implementations, which are not part of it.

Table 3: Median f32 solve time per class at tolerance 10^{-6}, over the instances that reached the tolerance and excluding compilation. Factors in parentheses are relative to explicit f32.

Class	implicit (ours)	explicit	cuClarabel	BoxOSQP
random QP	8.3 ms (2.3×)	3.6 ms	37.3 ms (10.4×)	726.7 ms (204×)
portfolio	177.2 ms (3.4×)	52.0 ms	73.7 ms (1.4×)	1405.6 ms (27×)
lasso	154.9 ms (2.2×)	71.9 ms	53.1 ms (0.7×)	2394.3 ms (33×)
SVM	396.4 ms (5.4×)	73.5 ms	74.0 ms (1.0×)	3839.5 ms (52×)
control	50.0 ms (2.4×)	20.6 ms	44.3 ms (2.1×)	835.3 ms (41×)

E.3 The benchmark under TF32

Sections 6.2–6.4 run under TF32, the arithmetic that a GPU applies to f32 matrix products by default (Appendix D). To connect the two settings, we reran the implicit and explicit solvers with the code of the supplementary material on the RTX 5090, once in true single precision and once under TF32, with at most 30 interior-point iterations, seed 0, and the success test of Appendix E.1. Table 4 reports the aggregate counts. In true single precision, the implicit counts reproduce Table 2, and the explicit counts move by a few instances at the tolerance boundary (500, 287, and 160 against 496, 272, and 154), the run-to-run variation of a benchmark whose GPU kernels are not deterministic. Under TF32, the implicit solver still reaches 10^{-3} on all 500 problems, and on portfolio, lasso, and control it reaches 10^{-5} and 10^{-6} exactly as in true single precision, whereas the explicit baseline drops to 310/500 already at 10^{-3}. On the two dense classes, random QP and SVM, the KKT residual of both solvers floors at about 3×10^{-4} on every instance (Table 5): this is the input rounding of TF32 acting on dense problem data, and no interior-point formulation can pass a 10^{-5} test through it. Every returned solution of both solvers is finite. Solve times are not reported for this run, which shared the GPU with other jobs.

Table 4: Successful solves out of 500 OSQP-benchmark QPs for the two solvers of the paper, rerun in true single precision (f32) and under TF32; re-verified in f64 as in Table 2.

	implicit (ours)		explicit	
tol	f32	TF32	f32	TF32
10^{-3}	500	**500**	500	**310**
10^{-5}	500	**299**	287	**113**
10^{-6}	497	**296**	160	**100**

Table 5: Per-class results under TF32: successful solves out of 100 (explicit / implicit) at each tolerance, and the median KKT residual attained by the implicit solver at 10^{-6} under TF32 and in true single precision.

Class	10^{-3}	10^{-5}	10^{-6}	implicit residual, TF32 / f32
random QP	41 / 100	0 / 0	0 / 0	3.0×10^{-4} / 1.5×10^{-7}
portfolio	100 / 100	0 / 99	0 / 99	2.3×10^{-7} / 2.5×10^{-7}
lasso	100 / 100	100 / 100	100 / 100	1.9×10^{-7} / 1.9×10^{-7}
SVM	1 / 100	0 / 0	0 / 0	3.8×10^{-4} / 1.4×10^{-7}
control	68 / 100	13 / 100	0 / 97	3.4×10^{-7} / 3.3×10^{-7}

E.4 Empirical conditioning of the reduced KKT system

To complement the limiting argument of Section 4, we measure the conditioning of the complete reduced KKT matrix along the interior-point trajectory on 150 benchmark QPs (the five classes above, three sizes, ten instances each), solved in f64 at tolerance 10^{-5}. Both formulations reduce to a symmetric three-block system of the same size $N = n + m + p$ (110–4531 here), so their condition numbers are directly comparable. For each solve we record the degradation during the solve, $g_{\mathrm{cond}} = \log_{10}(\max_k \mathrm{cond}_2(K_k)/\mathrm{cond}_2(K_0))$, and the worst value, $c_{\max} = \log_{10} \max_k \mathrm{cond}_2(K_k)$, where K_k denotes the reduced matrix at iteration k. Table 6 reports medians and 95th percentiles over instances. The replay that records these condition numbers is included in the supplementary material; it was run in f64 on the RTX 5090, where the arithmetic mode of Appendix D does not apply.

Table 6: Conditioning of the reduced KKT matrix along the interior-point trajectory (median [95th percentile] over instances). g_{cond} is the growth of the condition number during the solve and $c_{\max}$ its worst value, both in orders of magnitude.

	g_{cond}		$c_{\max}$	
Class	implicit	explicit	implicit	explicit
random QP	2.04 [3.24]	8.13 [9.07]	4.34 [5.56]	9.86 [10.95]
portfolio	3.29 [3.50]	9.91 [10.40]	5.22 [5.54]	11.66 [12.18]
lasso	1.42 [1.65]	8.15 [9.17]	3.30 [3.72]	10.14 [11.17]
SVM	1.36 [2.93]	8.13 [9.35]	3.33 [4.86]	9.08 [10.32]
control	0.04 [0.32]	7.29 [8.76]	3.92 [5.35]	11.05 [12.70]

Three observations follow. First, the separation is systematic rather than class-specific: the explicit formulation degrades by 7.3–9.9 orders of magnitude during the solve, against 0.04–3.3 for the implicit one. Second, the absolute levels matter in low precision: the implicit system peaks at $10^{3.3}$–$10^{5.2}$, leaving margin even in f32 (unit roundoff $\approx 10^{-7}$), whereas the explicit system peaks at $10^{9.1}$–$10^{11.7}$, beyond what f32 can absorb, which is the failure mode behind the explicit f32 success rates above. Third, on the control class the implicit growth is essentially zero: the conditioning is set by the initialization and does not deteriorate as the barrier parameter vanishes, which is precisely what the limiting argument predicts. No solve of either formulation failed or went singular in f64.

F Gradient Fidelity and Robustness in Single-Precision

F.1 Problem Formulation

Let

$$\mathcal{P} = \{y \in \mathbb{R}^n : Gy \leq h\} \tag{24}$$

be a polytope defined by p linear inequalities, where $G \in \mathbb{R}^{p \times n}$ and $h \in \mathbb{R}^p$. For a query point $x \in \mathbb{R}^n$, the experiment studies the Euclidean projection

$$\Pi_{\mathcal{P}}(x) \;=\; \arg\min_{y \in \mathbb{R}^n} \frac{1}{2}\|y - x\|_2^2 \quad \text{s.t.} \quad Gy \leq h. \tag{25}$$

This problem is encoded into the QPs with the following problem data

$$Q = I_n, \qquad c = -x, \qquad A_{\mathrm{eq}} \in \mathbb{R}^{0 \times n}, \qquad b_{\mathrm{eq}} \in \mathbb{R}^0, \tag{26}$$

so the objective solved in standard QP form is

$$\frac{1}{2}y^\top y - x^\top y, \tag{27}$$

which differs from (25) only by the constant $-\frac{1}{2}\|x\|_2^2$.

The quantity of interest is the gradient of the projection map with respect to the query point. Rather than comparing full Jacobian matrices, we benchmark the output with a fixed probe direction $v \in \mathbb{R}^n$ and differentiate

$$\phi(x) = v^\top \Pi_{\mathcal{P}}(x). \tag{28}$$

For each sampled instance, we compute

$$g(x) = \nabla_x \phi(x) \tag{29}$$

through automatic differentiation of the QP solver.

F.2 Exact Hard-Projection Reference

The reference gradient is constructed analytically from the active set. The generator builds instances so that the true projection point is a prescribed point y_0 on a face of the polytope with exactly m active constraints. Let $G_A \in \mathbb{R}^{m\times n}$ denote the corresponding active rows.

Under the fixed-active-set model, the local projection map is the orthogonal projection onto the affine subspace

$$\{y : G_A y = h_A\}, \tag{30}$$

whose Jacobian is defined as

$$J_{\text{hard}} = I_n - G_A^\top (G_A G_A^\top)^{-1} G_A. \tag{31}$$

Equation (31) represents the orthogonal projector onto the tangent space of the active face. Since J_{hard} is symmetric, the exact gradient of the scalarized objective $\phi(x) = v^\top \Pi_{\mathcal{P}}(x)$ is given by

$$g_{\text{hard}} = J_{\text{hard}} v. \tag{32}$$

F.3 Experiment design

To study how geometry influences gradient quality, we vary the location of the query point relative to the polytope boundary. More precisely, we control how far the point lies outside the polytope while keeping the projected point on the same active face. This isolates the effect of moving away from the boundary along the outward normal directions.

For each instance, we first sample a boundary point y_0 and a set of m active constraints, collected in the matrix G_A. We then define the query point as

$$x = y_0 + G_A^\top \lambda, \qquad \lambda = d\,\xi, \tag{33}$$

where the components of ξ are sampled independently as $\xi_i \sim \text{Uniform}(0.5, 1.5)$ and $d \in \text{logspace}(-2, 2, 21)$. In this construction, the scalar d controls the overall displacement away from the boundary point y_0. As d increases, the query point moves farther outward from the active face, and both the constraint violation and the distance $\|x - y_0\|_2$ increase approximately linearly.

Across all experiments we maintain the following fixed:

$$\text{solver_max_iter} = 100, \tag{34}$$

$$\text{seed} \in \{0, 1, 2\}. \tag{35}$$

Sweep over problem size. We fix $\kappa_{\text{relax}} = 10^{-4}$ and sweep the following:

$$n \in \{2, 6, 10, 14, 18, 22, 26, 30, 34, 38\}, \tag{36}$$

$$p = 1.25n, \tag{37}$$

$$m_r \in \{0.2, 0.6, 0.8\}, \tag{38}$$

$$\text{solver_tol} = 10^{-4}. \tag{39}$$

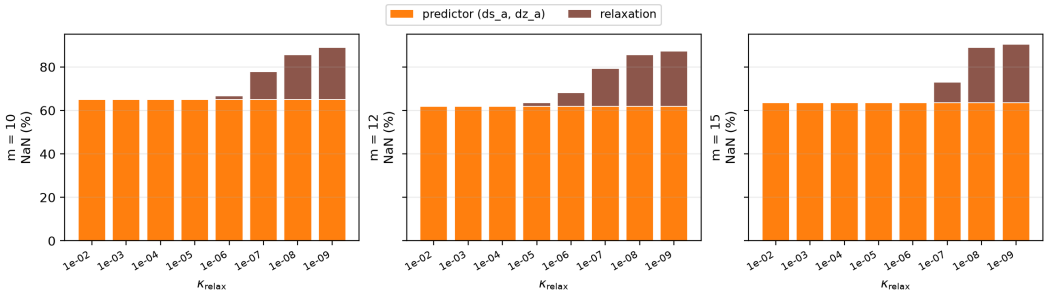

Figure 4: NaN tracing for operations in explicit f32 (TF32).

Sweep over relaxation parameter. We fix $n = 20, p = 25, m = \{10, 12, 15\}$ and sweep over relaxation parameter as:

$$\kappa_{\text{relax}} \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}\} \tag{40}$$

$$\text{solver_tol} = \min(\kappa_{\text{relax}}, 10^{-4}) \tag{41}$$

The solver tolerances above apply to f32, where tighter residuals are not meaningful given the unit roundoff of $\approx 10^{-7}$; f64 runs use $\text{solver_tol} = 10^{-8}$. The tolerance thus depends on the arithmetic precision, not on the formulation. Both sweeps run under TF32 (Appendix D).

F.4 NaN tracing in explicit f32

For the κ sweep in Fig. 1, under the same TF32 arithmetic, we trace the exact operations where the first non-finite intermediate appears inside f32 explicit operations.

Over the full sweep, first-hit failures are attributed only to the affine predictor step of the primal-dual interior-point method or to the subsequent relaxed continuation; all other traced groups remain inactive. The predictor accounts for 41, 39, and 40 of the 63 evaluations for $m = 10$, 12, and 15, independently of κ, whereas the relaxation adds failures only at tight relaxations, first at $\kappa = 10^{-6}$, 10^{-5}, and 10^{-7}, respectively, and reaching 15 to 17 cases at $\kappa = 10^{-9}$. The tracing replays each solve at the Python level rather than in the jitted run, so its totals fall a few cases short of the NaN rates of Fig. 1 at the tightest relaxations (55 to 57 of 63 at $\kappa = 10^{-9}$), and repeated launches move individual counts by one to three cases.

In particular we trace the following operations in the codebase of [7]. First, in the forward primal-dual interior-point solve, we separately test the KKT scaling vector z/s, the affine predictor directions (ds_a, dz_a), the centering quantities (σ, μ), the corrector directions (dx, ds, dz, dy), and the line-search step α. Then, we also check the relaxed solve and the custom backward solve. The results are given in the table 7 and plotted in Fig. 4.

G Bilevel Trajectory Optimization

G.1 Problem Formulation

G.1.1 Bézier curve parameterization

Let K denote the number of convex safe cells, and let each trajectory segment be a degree-n Bézier curve in $\mathbb{R}^d$. For our sweep experiments, we use $n = 4$ and $d = 2$. Each segment k is parameterized over a normalized local coordinate $u \in [0, 1]$ as:

$$p_k(u) = \sum_{j=0}^{n} P_{k,j} b_j^n(u), \tag{42}$$

Table 7: Operation-level first-hit nan tracing. Each row summarizes 63 instances. Column abbreviations are: Sc. (scaling), Pred. (predictor), Cent. (centering), Corr. (corrector), LS (line search), Rel. (relaxation), Bwd. (backward), and N/A (no attribution).

m	κ	total	Sc.	Pred.	Cent.	Corr.	LS	Rel.	Bwd.	N/A
10	10^{-2}	63	0	41	0	0	0	0	0	0
10	10^{-3}	63	0	41	0	0	0	0	0	0
10	10^{-4}	63	0	41	0	0	0	0	0	0
10	10^{-5}	63	0	41	0	0	0	0	0	0
10	10^{-6}	63	0	41	0	0	0	1	0	0
10	10^{-7}	63	0	41	0	0	0	8	0	0
10	10^{-8}	63	0	41	0	0	0	13	0	0
10	10^{-9}	63	0	41	0	0	0	15	0	0
12	10^{-2}	63	0	39	0	0	0	0	0	0
12	10^{-3}	63	0	39	0	0	0	0	0	0
12	10^{-4}	63	0	39	0	0	0	0	0	0
12	10^{-5}	63	0	39	0	0	0	1	0	0
12	10^{-6}	63	0	39	0	0	0	4	0	0
12	10^{-7}	63	0	39	0	0	0	11	0	0
12	10^{-8}	63	0	39	0	0	0	15	0	0
12	10^{-9}	63	0	39	0	0	0	16	0	0
15	10^{-2}	63	0	40	0	0	0	0	0	0
15	10^{-3}	63	0	40	0	0	0	0	0	0
15	10^{-4}	63	0	40	0	0	0	0	0	0
15	10^{-5}	63	0	40	0	0	0	0	0	0
15	10^{-6}	63	0	40	0	0	0	0	0	0
15	10^{-7}	63	0	40	0	0	0	6	0	0
15	10^{-8}	63	0	40	0	0	0	16	0	0
15	10^{-9}	63	0	40	0	0	0	17	0	0

where $P_{k,j} \in \mathbb{R}^d$ are the control points and b_j^n are the Bernstein polynomials. The physical time duration assigned to segment k is $T_k > 0$, yielding a total traversal time of:

$$T_{\text{tot}} = \sum_{k=1}^{K} T_k. \tag{43}$$

To compute trajectory derivatives efficiently, for each derivative order r, we construct a differentiation matrix M_r such that the control points of the r-th derivative with respect to the normalized parameter u are simply $M_r P_k$. When mapping back to physical time τ, the derivative must be scaled by T_k^{-r}:

$$\frac{d^r p_k}{d\tau^r}(\tau) = \frac{1}{T_k^r} \frac{d^r p_k}{du^r}(u). \tag{44}$$

This temporal scaling is the core source of the bilevel coupling in our formulation: both the quadratic cost matrix and several of the kinematic constraints in the inner problem depend non-linearly on the duration vector T.

G.1.2 Inner level: quadratic program over control points

Given a fixed segment duration vector T, we formulate the inner optimization problem by stacking all control points into a single decision vector $x \in \mathbb{R}^{K(n+1)d}$. The resulting trajectory optimization

problem is a purely quadratic program (QP):

$$\begin{aligned} \phi(T) = \min_{x} \quad & \frac{1}{2} x^\top Q(T) x \\ \text{s.t.} \quad & A_{\text{eq}}(T) x = b_{\text{eq}}(T), \\ & Gx \leq h(T). \end{aligned} \tag{45}$$

Note that the linear objective term is zero in our implementation, so the objective is entirely quadratic.

Objective Function. The cost matrix $Q(T)$ is block-diagonal across the trajectory segments and is designed to penalize the derivatives of the curve. For a given segment k, derivative order r, and penalty weight w_r, the continuous-time cost is:

$$\frac{w_r}{T_k^{2r-1}} \int_0^1 \left\| \frac{d^r p_k}{du^r}(u) \right\|^2 du. \tag{46}$$

Using Bernstein Gram matrices, this integral is evaluated exactly and assembled as a quadratic form over the control points. For our sweep experiments, we penalize velocity, acceleration, jerk, and snap using the weights:

$$(w_1, w_2, w_3, w_4) = (0.1,\ 0.1,\ 1.0,\ 0.1). \tag{47}$$

This configuration heavily penalizes jerk while applying milder regularization to the other derivatives.

The equality and inequality constraints in (45) enforce safety, continuity, and boundary conditions as follows:

Cell-membership constraints. We restrict the trajectory to safe regions [3] defined by convex polytopes$\mathcal{C}_k = \{p \in \mathbb{R}^2 \mid A_k p \leq b_k\}$. By constraining every Bézier control point of segment k to lie within $\mathcal{C}_k$, the convex-hull property of Bézier curves guarantees that the entire segment remains safely within the cell.

Endpoint constraints. The initial control point of the first segment is fixed to the designated start state, and the final control point of the last segment is fixed to the goal state.

Continuity constraints. To ensure the physical trajectory is C^2 continuous, adjacent segments must match up to the second derivative ($c = 2$). For $r \in \{0, 1, 2\}$, we enforce:

$$\frac{(M_r P_k)_{\text{end}}}{T_k^r} = \frac{(M_r P_{k+1})_{\text{start}}}{T_{k+1}^r}. \tag{48}$$

Initial and terminal derivative constraints. The trajectory must start and end at rest. We enforce zero velocity and acceleration at the boundaries for $r \in \{1, 2\}$:

$$\frac{(M_r P_1)_{\text{start}}}{T_1^r} = 0, \qquad \frac{(M_r P_K)_{\text{end}}}{T_K^r} = 0. \tag{49}$$

Derivative bounds. We impose strict velocity and acceleration limits by bounding the derivative control polygons. Because the derivative of a Bézier curve is itself a Bézier curve, constraining its control points strictly bounds the derivative along the entire segment. For $r \in \{1, 2\}$, we enforce:

$$\ell_r \leq \frac{M_r P_k}{T_k^r} \leq u_r. \tag{50}$$

In our sweep setup, the velocity limits are $\ell_1 = (-10, -10)$ and $u_1 = (10, 10)$, while the acceleration limits are $\ell_2 = (-2, -2)$ and $u_2 = (2, 2)$.

[3] We obtain the convex safe regions by using the python package pydecomp.

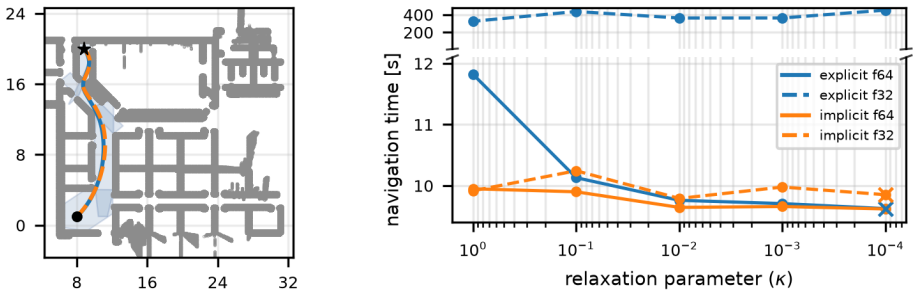

Figure 5: Bilevel optimization for fast (outer opt.) and safe (inner opt.) trajectories. *Left:* Trajectories in office environment, explicit f64 and implicit f32 at $\kappa_{\text{relax}} = 10^{-4}$. *Right:* Navigation time for trajectories computed with different relaxations; the upper panel holds explicit f32. Crosses in the right column plot refer to the trajectories picked for plotting in the left column.

G.1.3 Outer level: optimization over segment durations

The outer optimization seeks to find the optimal segment durations by augmenting the inner smoothness cost $\phi(T)$ with a linear time penalty:

$$\min_{T \in \mathbb{R}^K_{>0}} \mathcal{F}(T) := \phi(T) + \lambda \sum_{k=1}^{K} T_k \quad \text{s.t.} \quad T_k \geq T_{\min}. \tag{51}$$

In our experiments, we set $\lambda = 100$ and $T_{\min} = 0.1$. Without this linear time penalty, the optimization would naturally favor arbitrarily large durations to minimize the smoothness cost. By penalizing time, the problem becomes genuinely bilevel: reducing the duration T_k decreases the linear penalty but simultaneously tightens the feasible set and alters the objective curvature of the inner QP through the T_k^{-r} and $T_k^{-(2r-1)}$ scaling factors.

To solve the outer problem, we use L-BFGS [68] to compute a search direction, followed by an Armijo backtracking line search [33]. If the L-BFGS direction fails to yield a valid descent direction, the solver falls back to standard steepest descent.

G.2 Experimental design

The sweep uses the following problem and optimizer settings:

Quantity	Value
Bézier segment degree n	4
Continuity order c	2
Smoothness weights	$[0.1, 0.1, 1.0, 0.1]$
Boundary conditions	Zero initial/terminal velocity & acceleration
Derivative bounds	Specified for orders 1 and 2
Initial duration T_k	3.0
Time penalty λ	100
L-BFGS iterations	35
L-BFGS memory	10
Armijo constant c_1	10^{-4}
Backtracking factor	0.5
Maximum backtracking steps	30
Duration floor $T_{\min}$	0.1
Step stopping tolerance	10^{-8}

The solver settings and the relaxation parameters used in the sweep, identical for all four solver variants, are

$$\texttt{solver_tol} = 10^{-5}, \qquad \texttt{max_iter} = 100, \tag{52}$$

$$\kappa_{\text{relax}} = \{10^{0}, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}. \tag{53}$$

The L-BFGS loop declares an inner QP primal infeasible, and rejects the step, when its constraint violation exceeds $\max(10^{-3}, 10^{3}\,\texttt{solver_tol}) = 10^{-2}$. The f32 runs use TF32 (Appendix D).

G.3 Complementary Results

In addition to the results in Fig. 2, we provide additional qualitative visualizations of the bilevel trajectory optimization operating in an office environment in Fig. 5. Table 8 lists the navigation times behind both figures. Explicit f32 produces no usable result in either environment: at $\kappa_{\text{relax}} = 10^{-3}$ in the forest its first gradient is not finite, and everywhere else the line search fails within a few iterations, or the full 35 iterations make no progress, leaving the navigation time close to the inflated initial guess. Implicit f32 reaches the navigation time of the f64 variants at every relaxation in the office and at every relaxation but the loosest in the forest.

Runs that complete all 35 L-BFGS iterations reproduce to four or more digits; runs that stop early on a failed line search do not, in f64 as well as in f32. Explicit f64 at $\kappa_{\text{relax}} = 10^{-1}$ in the forest gave 13.56 s in one launch and 19.27 s in another with identical code, and implicit f32 at $\kappa_{\text{relax}} = 1$ gave 13.69, 18.16, and 14.41 s in three launches. The table and the figures keep the values of the launch that produced the shipped data.

Table 8: Navigation time in seconds for the sweeps of Fig. 2 (forest) and Fig. 5 (office). The number of completed L-BFGS iterations is given in parentheses when fewer than 35; "none" means that no iteration completed because the first gradient was not finite.

	forest				office			
κ_{relax}	impl. f64	impl. f32	expl. f64	expl. f32	impl. f64	impl. f32	expl. f64	expl. f32
10^{0}	13.71 (21)	18.16 (14)	17.91 (14)	112.9 (24)	9.95 (12)	9.92 (15)	11.82 (3)	325.7 (8)
10^{-1}	13.53	13.64 (28)	19.27 (10)	135.4 (6)	9.90 (19)	10.24 (12)	10.13 (9)	435.9 (4)
10^{-2}	13.92	13.64 (25)	13.92	91.6	9.65	9.79 (29)	9.76	362.8 (9)
10^{-3}	13.71	13.67 (33)	13.71	none	9.66	9.98 (15)	9.71	363.3 (8)
10^{-4}	13.69	13.73	13.69	143.5 (5)	9.62	9.85	9.63	451.7 (2)

H Learning a Safety Filter

H.1 Problem formulation

We consider n planar agents with positions $p_i \in \mathbb{R}^2$ and stacked state and control

$$x = \begin{bmatrix} p_1^\top & \cdots & p_n^\top \end{bmatrix}^\top \in \mathbb{R}^{2n}, u = \begin{bmatrix} u_1^\top & \cdots & u_n^\top \end{bmatrix}^\top \in \mathbb{R}^{2n}, \qquad p_{i,t+1} = p_{i,t} + \Delta t\, u_{i,t}. \tag{54}$$

Each agent must reach a prescribed goal while avoiding static obstacles and pairwise collisions. The learned controller predicts a nominal control $u^{\text{nom}}(x)$ together with state-dependent barrier gains $\alpha(x)$. These quantities parameterize a QP whose solution is the executed control $u^{\text{safe}}(x)$. The neural network therefore captures task intent, while the QP acts as a safety-preserving projection that modifies the nominal action only when constraints become active.

H.2 Safety filter as a CBF

Let m denote the number of circular obstacles, with center c_o and radius r_o. Obstacle avoidance is encoded by

$$h_{i,o}^{\text{obs}}(x) = \|p_i - c_o\|_2^2 - r_o^2. \tag{55}$$

For pairwise safety, with agent radius r_{agent},

$$h_{i,j}^{\text{pair}}(x) = \|p_i - p_j\|_2^2 - (2r_{\text{agent}})^2. \tag{56}$$

Under single-integrator dynamics, each barrier is enforced through the zeroing-CBF condition

$$\nabla h(x)^\top u + \alpha(x)\, h(x) \geq 0, \tag{57}$$

with learned positive gain $\alpha(x)$. The safety filter is the centralized QP

$$\begin{aligned} u^{\mathrm{safe}}(x) = \arg\min_{u \in \mathbb{R}^{2n}} \ & \frac{1}{2}\|u - u^{\mathrm{nom}}(x)\|_2^2 \\ \text{subject to } & \nabla_{p_i} h_{i,o}^{\mathrm{obs}}(x)^\top u_i + \alpha_{i,o}(x)\, h_{i,o}^{\mathrm{obs}}(x) \geq 0, \quad \forall i, o, \\ & \nabla h_{i,j}^{\mathrm{pair}}(x)^\top u + \alpha_{i,j}(x)\, h_{i,j}^{\mathrm{pair}}(x) \geq 0, \quad \forall i < j, \\ & -u_{\max} \leq u_k \leq u_{\max}, \quad k = 1, \ldots, 2n. \end{aligned} \tag{58}$$

This formulation is minimally invasive: it returns the closest safe control to the nominal proposal. Since pairwise constraints couple multiple agents, the QP is solved jointly over all controls.

H.3 Data generation

Training data are generated by rolling out an oracle safety filter over randomly sampled obstacle configurations. At each time step, an unconstrained goal-seeking controller proposes

$$u_i^{\mathrm{pd}}(x) = k_p \frac{g_i - p_i}{\max(\|g_i - p_i\|_2, \varepsilon)}, \tag{59}$$

which is clipped to the control bounds and passed through the oracle CBF-QP with fixed barrier gain. The executed safe action is stored together with the current state and obstacle layout. This yields a supervised dataset of dynamically relevant states, rather than arbitrary random samples, with concentration near regions where safety constraints become active.

H.4 Training

Training is performed by imitation learning. Given oracle-safe actions $u_\star^{(k)}$, we minimize

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{k=1}^{N} \left\| \pi_\theta(x^{(k)}, \mathcal{O}^{(k)}) - u_\star^{(k)} \right\|_2^2, \tag{60}$$

where π_θ is the learned controller and $\mathcal{O}^{(k)}$ denotes the obstacle layout for the k-th sample. We use Adam [70] with learning rate 10^{-3} for 50 epochs and batch size 256. In the nominal experiment, $\Delta t = 0.1$, $u_{\max} = 2.0$, the goal tolerance is 0.1, there are 3 obstacles, and $r_{\mathrm{agent}} = 0.35$. Training is run for every n from 2 to 9 agents, with 400 training environments, 40 test environments, and rollouts of length up to 200. For $n = 7$, the centralized QP has 14 variables and

$$4n + nm + \frac{n(n-1)}{2} = 28 + 21 + 21 = 70 \tag{61}$$

constraints. We compare two differentiable QP backends, implicit and explicit, each in float64 and float32 precision, together with a neural baseline without a QP layer.

H.5 Evaluation/Validation

Validation is performed by closed-loop rollout on unseen obstacle configurations. We measure obstacle violations, pairwise violations, and whether all agents reach their goals within the horizon. At every agent count, the successful CBF-QP models achieve zero safety violations on held-out rollouts, whereas the unconstrained neural baseline exhibits frequent collisions and much weaker task completion. Among the barrier-based variants, float64 and float32 implicit and float64 explicit train and validate successfully, while float32 explicit is numerically unstable in this regime. Table 9 lists the outcome for every n. The number of test configurations in which all agents reach their goals is the least stable metric: at $n = 9$, a second launch with identical f64 code reached 19 and 36 of 40 for the two f64 models, so differences of about ± 10 are training variance rather than a precision effect.

Table 9: Training outcome per agent count for the five models. Each cell gives the final-epoch training loss, the safety violations summed over the 40 test configurations, and the number of test configurations in which every agent ends within 0.5 of its goal. Explicit f32 returns NaN in the first batch of the first epoch at every n.

n	implicit f64	implicit f32	explicit f64	explicit f32	plain MLP
2	0.0090, 0, 39/40	0.0099, 0, 37/40	0.0100, 0, 39/40	NaN	0.043, 440, 28/40
3	0.059, 0, 40/40	0.060, 0, 39/40	0.057, 0, 40/40	NaN	0.073, 828, 21/40
4	0.018, 0, 40/40	0.017, 0, 40/40	0.018, 0, 40/40	NaN	0.055, 2077, 29/40
5	0.032, 0, 40/40	0.032, 0, 40/40	0.034, 0, 40/40	NaN	0.076, 1366, 31/40
6	0.029, 0, 40/40	0.029, 0, 40/40	0.029, 0, 40/40	NaN	0.077, 2288, 15/40
7	0.036, 0, 40/40	0.036, 0, 40/40	0.036, 0, 40/40	NaN	0.088, 6531, 6/40
8	0.050, 0, 39/40	0.052, 0, 38/40	0.051, 0, 39/40	NaN	0.100, 8929, 15/40
9	0.032, 0, 23/40	0.032, 0, 33/40	0.032, 0, 32/40	NaN	0.058, 11146, 13/40

H.6 Maximum Batch size analysis

To assess scalability, we vary the number of agents from $n = 2$ to $n = 9$ and record the largest batch size that can be trained without out-of-memory failure. This study is carried out for implicit in both float64 and float32 precision. As expected, the feasible batch size decreases rapidly with n because the centralized QP grows in both dimension and constraint count, with the pairwise term scaling quadratically in the number of agents. Across the tested range, float32 supports consistently larger batches than float64, by roughly a factor of two. Because the implicit formulation trains reliably in float32, this additional memory headroom becomes usable without sacrificing numerical robustness. This measurement was taken on the NVIDIA GeForce RTX 3080 (12 GB) of the submitted manuscript and was not repeated with the camera-ready code.

H.7 Numerical Details

Throughout the experiment, the four QP layers share the same settings,

$$\texttt{solver_tol} = 10^{-5}, \qquad \kappa_{\text{relax}} = 10^{-4}, \qquad \texttt{max_iter} = 30, \tag{62}$$

and the f32 layers run under TF32 (Appendix D).

The specific configuration used for data generation, training and validation is summarized below:

Quantity	Value
Time step Δt	0.1
Control bound $u_{\max}$	2.0
Goal tolerance during data generation	0.1
Number of agents n	2 to 9
Number of obstacles m	3
Workspace	$[-1, 11]^2$
Agent radius r_{agent}	0.35
Oracle solver	`explicit f64`
Training obstacle configurations	400
Test obstacle configurations	40
Maximum rollout length	200 steps
Fixed oracle barrier gain `alpha_fixed`	1.0
PD gain k_p	2.5
Training epochs	50
Learning rate	10^{-3}
Batch size for every nominal model	256

H.8 Complementary results

We show random configurations obtained from learning the multi-agent safety filter for various numbers of agents and different QP solver configurations:

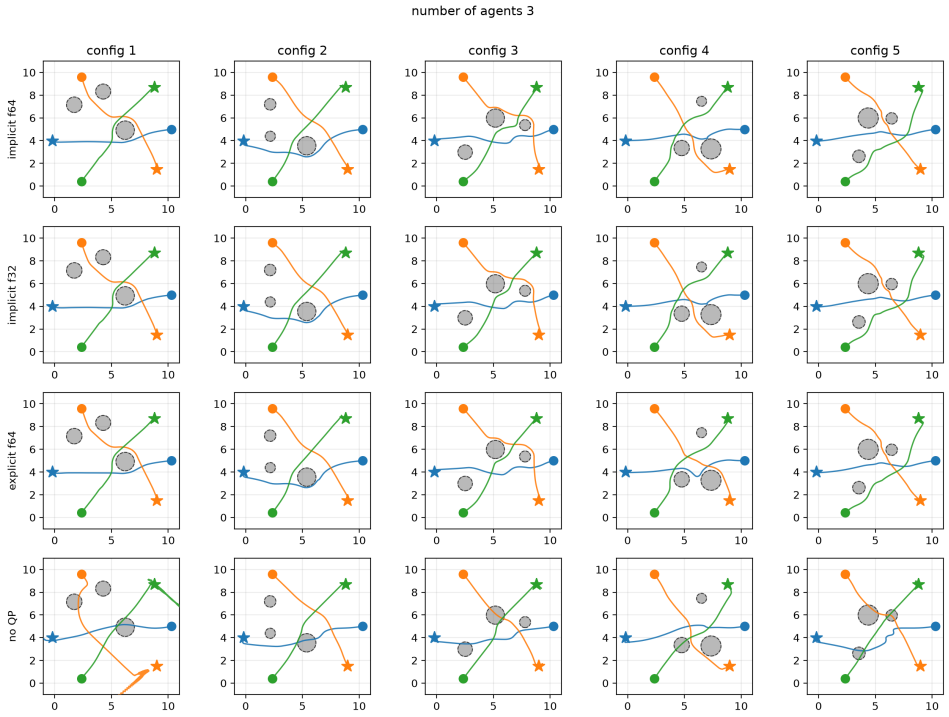

Figure 6: A subset of testing configurations for 3 agents.

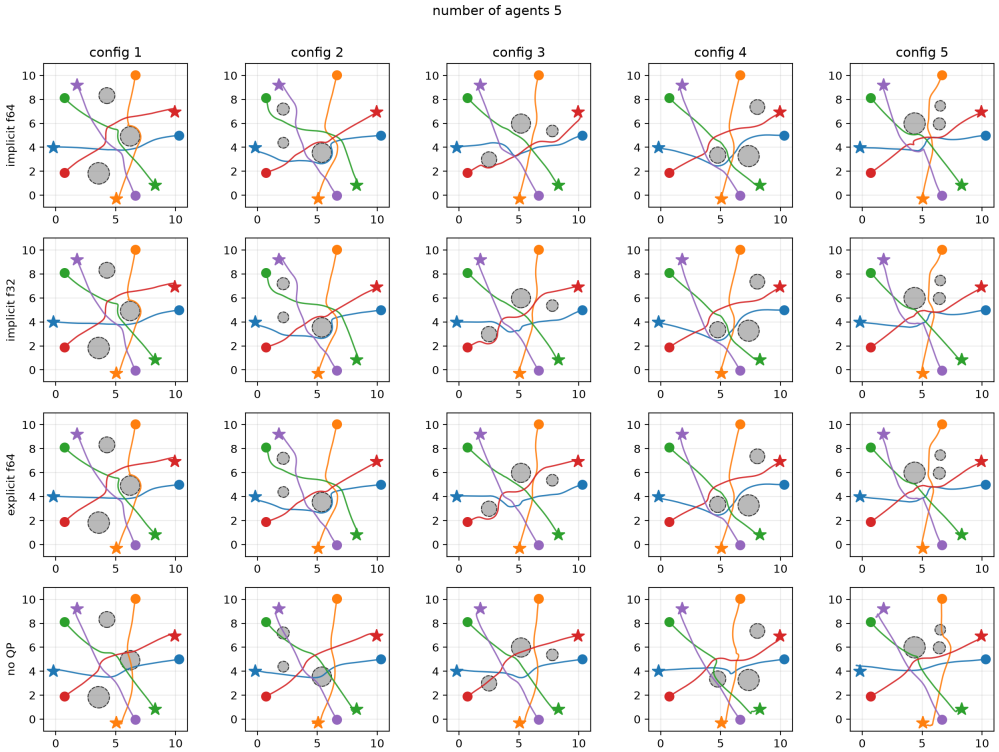

Figure 7: A subset of testing configurations for 5 agents.

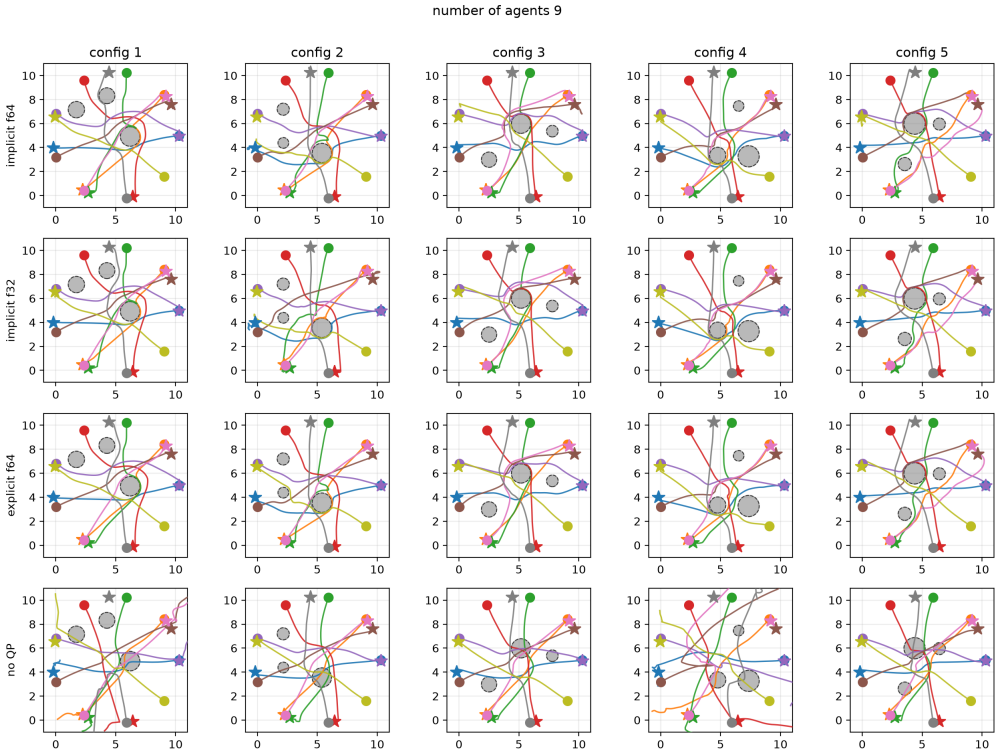

Figure 8: A subset of testing configurations for 9 agents.

I Extension to Other Symmetric Cones

The interior-point design of Algorithm 1 and the differentiation pipeline of Algorithms 2 and 3 carry over to other symmetric cones unchanged; only the retraction map b_κ is cone-dependent.

Second-order cones. The componentwise complementarity $z \odot s = \kappa \mathbf{1}$ is replaced, per cone block, by the Euclidean Jordan product $z \circ s = \kappa e$, where e is the identity element of the cone. Setting $z = b_\kappa(v)$ and $s = b_\kappa(-v)$ with

$$b_\kappa(v) = \tfrac{1}{2}\left(v + \sqrt{v \circ v + 4\kappa e}\right), \tag{63}$$

where the square root is taken in the Jordan algebra, amounts to applying the scalar map $\phi_\kappa(t) = \frac{1}{2}\left(t + \sqrt{t^2 + 4\kappa}\right)$ to the spectral values of v. This preserves $b_\kappa(v) - b_\kappa(-v) = v$ and $b_\kappa(v) \circ b_\kappa(-v) = \kappa e$, as in the QP case.

Semidefinite cones. The complementarity condition becomes $X \circ S = \frac{1}{2}(XS + SX) = \kappa I$. Setting $X = B_\kappa(V)$ and $S = B_\kappa(-V)$ with $V = X - S$ and

$$B_\kappa(V) = \tfrac{1}{2}\left(V + \sqrt{V^2 + 4\kappa I}\right), \tag{64}$$

where the principal matrix square root is used, amounts to applying ϕ_κ to the eigenvalues of V.

Cost. The representation is therefore cone-general, but its cost is not. For QPs (and LPs) the retraction is diagonal and costs $O(p)$; for second-order cones it is linear in the cone size but has a structured, non-diagonal Jacobian; for semidefinite cones it is spectral, costing $O(\sum_j n_j^3)$ over the PSD blocks. We have therefore treated the QP case as the foundation of this work: the efficient and well-conditioned instantiation of the implicit representation beyond QPs is not established here.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We provide analytical derivations and experimental results for all claims made in the abstract and contribution.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 is a dedicated Limitations section covering the computational cost of the implicit formulation, the scope of the conditioning argument, the restriction to feasible convex QPs, and the extension beyond QPs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in Appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Complete setup information for all experiments in Section 6 is provided in the Appendix. The source code required to reproduce our results is included in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code needed to reproduce the experimental results is attached in the supplementary material. The presented method is algorithmic and does not require any data.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Each experiment in Section 6 has an accompanying entry in the Appendix that provides all necessary information to understand the experimental design and selected hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in Appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For experiments with randomness, we report aggregated statistics rather than single-run outcomes, including medians over three random seeds in Fig. 1 and exact failure-rate/NaN-count statistics in Appendix F.4 (Table 7). These summaries are appropriate in our setting, where numerical instability produces discrete failures and strongly non-Gaussian variability.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources that we used are stated in Section 6 and Appendix D: the GPU of each experiment, the software version, and the arithmetic mode.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work performed in this paper.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The code used in the paper is properly credited and cited in Section 6. The OSQP benchmark problem generators are redistributed with our code under their Apache-2.0 license.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The code attached in the supplementary material is fully-documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.