LAPANDA: A MATRIX-FREE DIFFERENTIABLE SOLVER FOR NONCONVEX CONSTRAINED OPTIMIZATION LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Differentiable optimization brings the structural guarantees of mathematical optimization to network pipelines, allowing them to be trained end-to-end. However, its application remains challenging for nonconvex constrained problems, as existing differentiable solvers often suffer from limited modeling expressiveness due to their reliance on specialized problem structures, while also incurring substantial computation time and memory overhead in both the forward and backward passes. To address these challenges, we propose `lapanda`, a matrix-free differentiable solver for nonconvex optimization with general constraints. It reformulates the problem to a sequence of augmented Lagrangian subproblems, each handled by a first-order inner solver through a proximal averaged quasi-Newton algorithm with adaptive linesearch, thus enabling efficient forward optimization. We establish local well-posedness of the solution map and convergence of the outer iterations, and further derive a sensitivity alignment between the original problem and the final subproblem in the backward pass, demonstrating that the subproblem sensitivity, which can be computed efficiently in a matrix-free manner, provides a principled approximation to the exact optimizer sensitivity. We evaluate `lapanda` on nonconvex constrained Rosenbrock benchmarks, imitation learning with several representative constrained optimal control problems, and embedded robotic obstacle-avoidance tasks. Compared with state-of-the-art differentiable solvers, `lapanda` delivers substantial reductions in computation time and memory footprint while maintaining reliable constraint satisfaction and learning performance. Code: `https://anonymous.4open.science/r/lapanda_solver`

1 INTRODUCTION

Differentiable optimization serves as a bridge between optimization and machine learning, enabling structural priors and constraints to be embedded into learning frameworks (Kotary et al., 2021). This paradigm has been applied to a wide range of learning and control problems and safety-critical applications, such as learning personalized driving behaviors from demonstrations (Acerbo et al., 2023; 2024), integrating model-based planning into end-to-end autonomous driving systems (Huang et al., 2024a;b; Karkus et al., 2023), and combining reinforcement learning with optimal control for robotic systems (Romero et al., 2024; Wan et al., 2024; Xu & She, 2025).

Despite its promise, differentiable optimization faces two practical bottlenecks. First, many existing differentiable solvers gain computational efficiency by tailoring their formulations to specific problem structures, including convexification (Agrawal et al., 2019a), least-squares objectives (Pineda et al., 2022), and temporal structure in optimal control problems (OCPs) (Amos et al., 2018). Second, both optimization and sensitivity computation typically rely on explicit KKT matrix construction and factorization, resulting in substantial computational and memory overhead (Blondel et al., 2022). These bottlenecks are particularly restrictive for general nonconvex constrained problems with limited resources. We therefore propose `lapanda`, a matrix-free differentiable solver for nonconvex optimization layers with general constraints. The contributions are summarized as follows:

- **Efficient Optimization.** To enable efficient optimization under general constraints, `lapanda` combines the augmented Lagrangian method (ALM) with a first-order `PANDA` solver. ALM handles the constraints, while `PANDA` takes care of the resulting subproblems. We theoretically proved that the forward solution is locally well-posed and convergent.

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

- **Matrix-Free Differentiation.** To differentiate through the optimizer without constructing derivative matrices, we analyzed the alignment between the differential KKT systems of the original problem and the last ALM subproblem, then derived an error bound for approximating the exact solution sensitivity by the one of the final ALM subproblem. This leads to a backward pass based only on matrix-vector product operators.
- **Comprehensive Evaluation.** To evaluate `lapanda` from multiple perspectives, we considered three classes of optimization problems with distinct structures: nonconvex constrained Rosenbrock benchmarks, imitation learning via parametric constrained OCP, and embedded mobile robot obstacle-avoidance tasks. The results show that `lapanda` reliably completes the tasks while achieving competitive computational and memory efficiency.
- **Cross-Platform Software.** To facilitate practical deployment, we developed a C-based solver core with user-friendly Python and MATLAB interfaces, together with engineering optimizations that reduce cross-language interface overhead.

2 Related Work

Since `OptNet` introduced the idea of embedding a parametric optimization problem as a differentiable layer into a learning architecture (Amos & Kolter, 2017), a growing number of differentiable optimization solvers have been crafted. Many of these methods target specific problem classes. Examples include differentiable quadratic programming layers such as `qpth` and ADMM-based QP layers (Amos & Kolter, 2017; Butler & Kwon, 2023), disciplined convex and conic optimization layers such as `CVXPYLayers` and `diffcp` (Agrawal et al., 2019a;b), and nonlinear least-squares layers such as `Theseus` (Pineda et al., 2022). Despite their effectiveness, these specialized formulations make them less suitable for general nonconvex constrained optimization problems.

More general differentiable optimization frameworks are also available. For general nonlinear programs, `CasADi` combines interior-point optimization with active-set KKT differentiation (Andersson et al., 2019; Andersson & Rawlings, 2018). For general constrained OCPs, `acados` combines structure-exploiting SQP with smoothed interior-point sensitivities (Verschueren et al., 2022; Frey et al., 2025), while `SafePDP` computes sensitivities through a differentiable Pontryagin method (Jin et al., 2021). More recently, `TurboMPC` exploits GPU parallelism to accelerate differentiable model predictive control (MPC) (Bravo-Palacios et al., 2026). However, despite their broad applicability, these frameworks often explicitly construct and factorize matrices arising from KKT systems, imposing considerable computational and memory overhead, especially on resource-limited platforms.

A complementary line of work reduces this second-order burden. `FFOLayer` approximates hypergradients using only first-order information for convex programs, but requires additional forward optimization (Zhao et al., 2026). Building on the forward-backward envelope framework (Stella et al., 2017; Themelis et al., 2018), `PANDA` employs first-order iterations with adaptive stepsizes in the forward pass and matrix-free Krylov methods based on matrix-vector products in the backward pass, but supports only constraints that admit efficient proximal mappings (Chen et al., 2026).

Building on these ideas, `lapanda` extends the `PANDA` framework to general constraints. While prior work has combined ALM with proximal-type methods for forward optimization (Pas et al., 2022), we further establish local well-posedness and convergence for this formulation and exploit the subproblem structure to derive the sensitivity alignment and efficient matrix-free differentiation. Table 1 highlights the unique positioning of `lapanda` among representative differentiable solvers.

Table 1: Comparison of representative differentiable optimization solvers.

Solver	Target problem	Hard constraints	Matrix-free	Platform
lapanda (Ours)	General NLP	General	✓	C / MATLAB / Python
CasADi (2019)	General NLP	General	✗	C / MATLAB / Python
acados (2025)	General OCP	General	✗	C / MATLAB / Python
TurboMPC (2026)	General OCP	General	✗	Python
SafePDP (2021)	General OCP	General	✗	Python
PANDA (2026)	Composite NLP	Prox.-friendly	✓	MATLAB
FFOLayer (2026)	Convex	Convex	✓	Python
mpc.pytorch (2018)	OCP	Box	✗	Python
Theseus (2022)	NLS	–	✗	Python
qpth (2017)	Convex QP	Convex	✗	Python
CVXPYLayers (2019)	Convex	Convex	✗	Python

3 Problem Formulation

We consider a parametric constrained optimization problem of the form

$$\begin{aligned} u^\star(\theta) \in \underset{u \in \mathbf{U}}{\operatorname{argmin}} \quad & \ell(u, \theta) \\ \text{s.t.} \quad & c(u, \theta) = 0, \\ & h(u, \theta) \le 0, \end{aligned} \tag{P}$$

where $u \in \mathbb{R}^{n_u}$ is the decision variable and $\theta \in \mathbb{R}^{n_\theta}$ denotes the learnable parameter. The objective $\ell : \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \to \mathbb{R}$ and the constraint functions $c : \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_c}$ and $h : \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_h}$ are smooth and possibly nonconvex; c and h represent equality and inequality constraints, respectively. The set $\mathbf{U}$ represents a simple constraint set, such as a box, whose proximal mapping can be evaluated efficiently.

In differentiable optimization, the ***forward pass*** corresponds to solving (P) for a given parameter θ, yielding an optimizer $u^\star(\theta)$. In many learning problems, θ must also be adjusted so that the resulting optimizer minimizes a smooth task-level objective $\Phi(\theta)$. This leads to the outer learning problem

$$\min_{\theta} \quad \Phi(\theta) := \mathcal{L}_{\text{out}}\big(u^\star(\theta), \theta\big), \tag{1}$$

where $\mathcal{L}_{\text{out}}$ can be an imitation loss or performance loss. Differentiating the outer objective yields the chain rule as follows

$$\nabla_\theta \Phi(\theta) = \nabla_u \mathcal{L}_{\text{out}}\big(u^\star(\theta), \theta\big) \frac{\partial u^\star(\theta)}{\partial \theta} + \nabla_\theta \mathcal{L}_{\text{out}}\big(u^\star(\theta), \theta\big). \tag{2}$$

The main computational challenge in evaluating Eq. (2) lies in the optimizer sensitivity $\frac{\partial u^\star(\theta)}{\partial \theta}$, whose efficient computation constitutes the ***backward pass*** of (P).

Optimal control problems. Constrained optimal control problems are common in the control of autonomous systems. A typical discretized finite-horizon OCP has the form

$$\begin{aligned} \xi_\theta^\star = \{x_{0:N}^\star, u_{0:N-1}^\star\} \in \underset{\{x_k, u_k\}}{\operatorname{argmin}} \quad & \sum_{k=0}^{N-1} \ell_k(x_k, u_k; \theta) + \ell_N(x_N; \theta) \\ \text{s.t.} \quad & x_0 = x_{\text{init}}(\theta), \\ & x_{k+1} = f_k(x_k, u_k; \theta), \\ & c_k(x_k, u_k; \theta) = 0, \quad h_k(x_k, u_k; \theta) \le 0, \\ & c_N(x_N; \theta) = 0, \quad h_N(x_N; \theta) \le 0, \\ & u_k \in \mathbf{U}_k, \quad k = 0, \dots, N-1. \end{aligned} \tag{OCP}$$

Here, $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ denote the state and control, respectively, and $f_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$ denotes the dynamics. The functions ℓ_k, c_k and h_k are the stage-wise counterparts of ℓ, c and h in problem (P), with ℓ_N, c_N, and h_N denoting their terminal counterparts. While (OCP) naturally has a stage-wise dynamic structure, a single-shooting transcription (Gros & Diehl, 2024) recursively eliminates the states as decision variables, yielding a control-only constrained optimization problem in the generic form (P). This allows `lapanda` to be applied directly. We therefore develop the solver using (P) and defer the transcription details to Appendix A.

4 LAPANDA

This section details the algorithmic principle of `lapanda`, as shown in Figure 1. The ***forward pass*** is detailed in Section 4.1, where the nonconvex constrained problem is reformulated into ALM subproblems and reduced to a composite form handled by `PANDA`. In Section 4.2, we derive the ***backward pass*** by comparing the sensitivity systems of the original problem and the final ALM subproblem, thereby motivating a matrix-free approximate implementation.

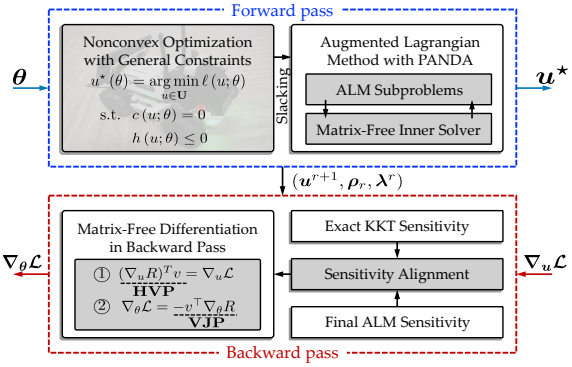


Figure 1: Overview of the lapanda framework.

Algorithm 1 Forward pass of lapanda

Input: parameter θ, initial primal variable u_0, multiplier λ_0, penalty ρ_0, constants $\kappa_{\mathrm{dec}} \in (0,1)$, $\tau_\rho > 1$, tolerances $\varepsilon_{\mathrm{ALM}} > 0$, and maximum number of ALM iterations $R_{\max} \in \mathbb{N}_+$
Set $z_0 = \Pi_{\mathcal{D}}(G(u_0,\theta) + \rho_0^{-1}\lambda_0)$ and $s_0 = G(u_0,\theta) - z_0$
for $r = 0, 1, \ldots, R_{\max} - 1$ **do**
  Solve the ALM subproblem $(u_{r+1}, z_{r+1}) \approx \arg\min_{u\in\mathbf{U},\ z\in\mathcal{D}} \mathcal{L}_{\rho_r}(u, z, \lambda_r; \theta)$
  Evaluate $s_{r+1} = G(u_{r+1},\theta) - z_{r+1}$
  Update $\lambda_{r+1} = \lambda_r + \rho_r s_{r+1}$
  Set $\rho_{r+1} = \rho_r$ if $\|s_{r+1}\| \le \kappa_{\mathrm{dec}}\|s_r\|$, and $\rho_{r+1} = \tau_\rho \rho_r$ otherwise
  if $\|s_{r+1}\|_\infty \le \varepsilon_{\mathrm{ALM}}$ **then**
    break
  end if
end for
Return: $u_{r+1} \approx u^\star(\theta)$

4.1 Forward Pass: Augmented Lagrangian with PANDA

We now describe the forward pass of `lapanda` for solving the parameterized problem (P). To make the following calculation and analysis well defined, we impose the standing assumptions below.

Assumption 1. *Let $u^\star$ be a local solution of problem (P). We assume that:*

A1. The functions $\ell(u,\theta)$, $c(u,\theta)$, and $h(u,\theta)$ are continuously differentiable and their derivatives are locally Lipschitz continuous. The simple set $\mathbf{U}$ is nonempty and closed.

A2. The functions $\ell(u,\theta)$, $c(u,\theta)$, and $h(u,\theta)$ are $C^{2,1}$ (twice continuously differentiable with locally Lipschitz second derivatives) in a neighborhood of $u^\star$. The set $\mathbf{U}$ admits a local $C^{2,1}$ active-constraint representation near $u^\star$. With this representation, the full constrained problem satisfies the linear independence constraint qualification (LICQ), the second-order sufficient condition (SOSC), and strict complementarity at $u^\star$.

Slack reformulation and augmented Lagrangian. Firstly, we rewrite the general equality and inequality constraints in a unified set-membership form by defining

$$G(u,\theta) := \begin{bmatrix} c(u,\theta) \\ h(u,\theta) \end{bmatrix}, \qquad \mathcal{D} := \{0\}^{n_c} \times \mathbb{R}_-^{n_h}.$$

Then the constraints in (P) can be compactly written as $G(u,\theta) \in \mathcal{D}$. By introducing an auxiliary slack variable $z \in \mathcal{D}$, problem (P) is equivalently written as

$$\begin{aligned} \min_{u\in\mathbf{U},\ z\in\mathcal{D}} \quad & \ell(u,\theta) \\ \text{s.t.} \quad & G(u,\theta) - z = 0. \end{aligned} \tag{3}$$

With a penalty parameter $\rho > 0$ and multiplier $\lambda \in \mathbb{R}^{n_c+n_h}$, its augmented Lagrangian is given as

$$\mathcal{L}_\rho(u, z, \lambda; \theta) = \ell(u,\theta) + \lambda^\top \big(G(u,\theta) - z\big) + \frac{\rho}{2}\|G(u,\theta) - z\|^2. \tag{4}$$

At the r-th ALM iteration, given (λ_r, ρ_r), the corresponding augmented Lagrangian subproblem reads

$$(u_{r+1}, z_{r+1}) \approx \operatorname*{argmin}_{u\in\mathbf{U},\ z\in\mathcal{D}} \mathcal{L}_{\rho_r}(u, z, \lambda_r; \theta). \tag{P-sub}$$

Then, the ALM forward pass proceeds as follows: (i) Solve (P-sub); (ii) Update the multiplier using the resulting feasibility residual; and (iii) Increase the penalty parameter only when the residual fails to decrease sufficiently. The complete calculation steps are summarized in Algorithm 1.

Remark 1 (Inner subproblem solved by PANDA)**.** *The main computational burden in Algorithm 1 arises from solving the ALM subproblem. Following the reduction in Pas et al. (2022), we eliminate z before invoking the inner solver, yielding a reduced problem only in u*

$$u_{r+1} \approx \operatorname*{argmin}_{u\in\mathbf{U}} \psi_{\rho_r}(u; \theta, \lambda_r), \tag{5}$$

where

$$\psi_\rho(u; \theta, \lambda) := \ell(u,\theta) + \frac{\rho}{2}\operatorname{dist}^2\left(G(u,\theta) + \rho^{-1}\lambda, \mathcal{D}\right). \tag{6}$$

Since $\mathbf{U}$ *typically represents simple control-input constraints, often in the form of box constraints, this reduced problem is equivalently written as the smooth-plus-nonsmooth composite problem*

$$\min_{u} \quad \psi_{\rho_r}(u;\theta,\lambda_r) + \delta_{\mathbf{U}}(u), \tag{7}$$

which can be handled efficiently in a matrix-free manner by `PANDA`*. The* z*-elimination derivation and the inner* `PANDA` *procedure are given in Appendix B.1 and Appendix B.2, respectively.*

Remark 2 (Warm start)**.** *Warm-starting plays a key role in the computational efficiency of* `lapanda`*. The initial primal variable, multiplier, and penalty parameter affect both the number of ALM outer iterations and the conditioning of the subproblem. An overly small penalty may require repeated penalty increases, whereas an overly large penalty can make the inner problem ill-conditioned. Thus, warm-starting with information from previous solves is an effective strategy.*

For the forward pass computation described above, we draw on local augmented Lagrangian theory (Bertsekas, 1999; Nocedal & Wright, 2006) to establish the following local well-posedness and convergence result.

Theorem 1 (Local well-posedness and convergence)**.** *Suppose Assumption 1 holds at a local solution* $(u^\star, z^\star, \lambda^\star)$ *of the slack formulation Eq. (3). Then there exist positive constants* $\bar{\rho}$*,* δ*,* ϵ*, and* M *such that, for any* λ_r *and* ρ_r *satisfying* $\|\lambda_r - \lambda^\star\| \le \rho_r\delta$ *and* $\rho_r \ge \bar{\rho}$*, the following statements hold:*

- *(i) The slack ALM subproblem (P-sub) is locally well defined. More precisely, it admits a unique local solution* (u_{r+1}, z_{r+1}) *in the* ϵ*-neighborhood of* $(u^\star, z^\star)$*. Moreover, the subproblem satisfies LICQ and SOSC at* (u_{r+1}, z_{r+1})*.*
- *(ii) With the multiplier update in Algorithm 1, the local primal-dual estimates satisfy*

$$\|(u_{r+1}, z_{r+1}) - (u^\star, z^\star)\| \le M\frac{\|\lambda_r - \lambda^\star\|}{\rho_r}, \qquad \|\lambda_{r+1} - \lambda^\star\| \le M\frac{\|\lambda_r - \lambda^\star\|}{\rho_r}. \tag{8}$$

- *(iii) After eliminating* z*, the reduced composite subproblem (7) satisfies the local* `PANDA` *regularity conditions at* u_{r+1}*. In particular,* u_{r+1} *is a strong local minimizer, and* $\delta_{\mathbf{U}}$ *is locally prox-regular relative to the identified active manifold. Consequently, the* `PANDA` *inner iteration is locally well posed and converges locally to* u_{r+1}*.*

Proof. See Appendix B.3. □

4.2 Backward Pass: Sensitivity Alignment and Matrix-Free Differentiation

This subsection derives the backward pass of `lapanda` through sensitivity alignment and matrix-free differentiation. As shown in the outer-gradient expression Eq. (2), the key computational quantity is the solution sensitivity $D_\theta u^\star$. We first characterize this sensitivity through the optimality conditions of the original constrained problem.

Exact sensitivity from the original KKT system. Under Assumption 1, strict complementarity implies local active-set identification. Let $\mathcal{A}_h$ be the active set of $h(u,\theta) \le 0$ at $u^\star$. We write the locally active smooth constraints as

$$a(u,\theta) := \begin{bmatrix} c(u,\theta) \\ h_{\mathcal{A}_h}(u,\theta) \end{bmatrix}, \qquad \phi(u) = 0, \tag{9}$$

where $\phi(u) = 0$ represents the identified active manifold of the simple set $\mathbf{U}$.

Let $A = \nabla_u a(u^\star,\theta)$, $C = \nabla_u \phi(u^\star)$, and let $\eta^\star, \mu^\star$ be the corresponding multipliers. The local active-set KKT system of Problem (P) is

$$\nabla_u \ell(u^\star,\theta) + A^\top\eta^\star + C^\top\mu^\star = 0, \qquad a(u^\star,\theta) = 0, \qquad \phi(u^\star) = 0. \tag{10}$$

Applying the implicit function theorem to Eq. (10) and differentiating with respect to θ gives

$$\begin{bmatrix} H & A^\top & C^\top \\ A & 0 & 0 \\ C & 0 & 0 \end{bmatrix} \begin{bmatrix} D_\theta u^\star \\ D_\theta \eta^\star \\ D_\theta \mu^\star \end{bmatrix} = -\begin{bmatrix} B \\ D \\ 0 \end{bmatrix}, \tag{11}$$

where $H := \nabla^2_{uu}(\ell + \eta^{\star\top} a + \mu^{\star\top}\phi)$, $\quad B := \nabla^2_{u\theta}(\ell + \eta^{\star\top} a + \mu^{\star\top}\phi)$, and $D := \nabla_\theta a$.

Eq. (11) characterizes the exact sensitivity required by the backward pass. Instead of explicitly forming and solving this saddle-point system, which would incur substantial memory overhead, we align it with the optimality conditions of the last ALM subproblem. Such alignment exposes the same composite structure exploited by `PANDA` in the forward solution pass, thereby enabling a matrix-free backward pass with substantially reduced memory requirements.

Sensitivity alignment through the final ALM subproblem. For the final ALM subproblem (P-sub), let η_r denotes the components of the incoming ALM multiplier λ_r associated with the equality and locally active inequality constraints. Define the corresponding shifted multiplier as $\bar{\eta}_{r+1} := \eta_r + \rho_r a(u_{r+1}, \theta)$. The local optimality conditions of the final reduced `lapanda` subproblem are

$$\nabla_u \ell(u_{r+1}, \theta) + A_{r+1}^\top \bar{\eta}_{r+1} + C_{r+1}^\top \mu_{r+1} = 0, a(u_{r+1}, \theta) - \rho_r^{-1}(\bar{\eta}_{r+1} - \eta_r) = 0, \phi(u_{r+1}) = 0, \tag{12}$$

where $A_{r+1} := \nabla_u a(u_{r+1}, \theta)$ and $C_{r+1} := \nabla_u \phi(u_{r+1})$.

When differentiating the final ALM subproblem, the incoming multiplier η_r and penalty parameter ρ_r are treated as fixed. Differentiating Eq. (12) therefore gives

$$\begin{bmatrix} H_{r+1} & A_{r+1}^\top & C_{r+1}^\top \\ A_{r+1} & -\rho_r^{-1} I & 0 \\ C_{r+1} & 0 & 0 \end{bmatrix} \begin{bmatrix} D_\theta u_{r+1} \\ D_\theta \bar{\eta}_{r+1} \\ D_\theta \mu_{r+1} \end{bmatrix} = - \begin{bmatrix} B_{r+1} \\ D_{r+1} \\ 0 \end{bmatrix}, \tag{13}$$

where $H_{r+1} := \nabla_{uu}^2 \left(\ell + \bar{\eta}_{r+1}^\top a + \mu_{r+1}^\top \phi\right), B_{r+1} := \nabla_{u\theta}^2 \left(\ell + \bar{\eta}_{r+1}^\top a + \mu_{r+1}^\top \phi\right)$ and $D_{r+1} := \nabla_\theta a$, with all derivatives evaluated at u_{r+1}. The detailed derivation of Eqs. (12)–(13) is provided in Appendix B.4. Comparing Eq. (13) with Eq. (11), the two systems differ only through the ALM regularization block $-\rho_r^{-1} I$ and the finite-iterate mismatch from the original primal-dual solution. This motivates the following theory.

Theorem 2 (Sensitivity alignment)**.** *Suppose the assumptions of Theorem 1 hold. Define*

$$\epsilon_{r+1} := \|u_{r+1} - u^\star\| + \|\bar{\eta}_{r+1} - \eta^\star\| + \|\mu_{r+1} - \mu^\star\|.$$

Then the sensitivity obtained from Eq. (13) satisfies

$$D_\theta u_{r+1} = D_\theta u^\star + O(\rho_r^{-1}) + O(\epsilon_{r+1}). \tag{14}$$

If the ALM outer iterates converge to the local primal-dual solution, then $\epsilon_{r+1} \to 0$, and the remaining discrepancy is $O(\rho_r^{-1})$. Moreover, if $\rho_r \to \infty$, then $D_\theta u_{r+1} \to D_\theta u^\star$.

Proof. See Appendix B.5. □

Theorem 2 explains that the sensitivity obtained by differentiating the converged ALM subproblem approximates the exact solution sensitivity with an explicit error bound. Under the residual viewpoint, without explicitly constructing the system Eq. (13), we compute $D_\theta u_{r+1}$ through the corresponding adjoint system using matrix-free Krylov methods and automatic-differentiation-based operators. Moreover, the penalty term in the subproblem yields locally positive curvature in the reduced space, which makes the efficient conjugate gradient (CG) method directly applicable. Appendix B.6 derives the required operator expressions and details the sensitivity computation procedure.

5 Experiments

We evaluate `lapanda` through three groups of experiments on representative nonconvex problems with general constraints, focusing on computational efficiency and sensitivity accuracy, learning performance, and embeddable implementation. For each experimental setting, we compare with representative solvers from Table 1 that support the corresponding problem structure.

5.1 Nonconvex Constrained Rosenbrock Benchmark

We first consider a nonconvex Rosenbrock-chain benchmark with smooth nonlinear constraints:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{i=0}^{n-2} \left[\theta_0 (x_{i+1} - x_i^2)^2 + \theta_1 (1 - x_i)^2\right] + \frac{1}{2}\theta_2 \|x\|_2^2 \\ \text{s.t.} \quad & -2 \le x_i \le 2, \qquad i = 0, \dots, n-1, \\ & g_i(x, \theta) = x_i^2 + x_{i+1}^2 - r_i(\theta)^2 \le 0, \qquad i = 0, \dots, n-2, \end{aligned} \tag{15}$$

where $r_i(\theta) = \theta_3 + 0.1 \sin\left(\frac{2\pi(i+1)}{n}\right)$. We vary the problem dimension n and compare `lapanda` with two baselines. `CasADi-IPOPT` is a widely used baseline for general nonlinear programs, while `Explicit KKT` explicitly forms the original problem's KKT sensitivity system in Eq. (11) and solves this indefinite saddle-point system using MINRES. With the relative gradient error kept below 2%, we compare their computation times and memory consumption.

Table 2: Matched-accuracy comparison on the constrained Rosenbrock benchmark.

n	Forward time (ms)			Backward time (ms)			Memory (MB)		
	lapanda	Explicit	CasADi	lapanda	Explicit	CasADi	lapanda	Explicit	CasADi
100	**8.23**	–	36.50	4.04	4.22	**3.04**	**1.78**	5.01	17.55
200	**14.37**	–	108.19	**5.89**	10.23	17.05	**2.06**	7.78	29.39
500	**49.75**	–	496.03	**14.36**	51.26	245.99	**3.12**	25.32	84.70
1000	**200.00**	–	1968.78	**37.36**	218.21	2325.16	**5.31**	79.65	249.66

The main results are reported in Table 2. `lapanda` is generally faster than both baselines, particularly for larger problem instances. The forward speedup mainly stems from the efficient `PANDA` inner solver, while the backward advantage comes from the lightweight operator-based implementation. Its matrix-free formulation also provides a clear memory advantage. In the representative case shown in Fig. 2, most nonlinear constraints are active and are handled well by the augmented Lagrangian method of `lapanda`. Appendix C.1 details the experimental settings, analyzes the sensitivity-error behavior predicted by Theorem 2, and describes both the strategies used to control this error and the accuracy-matching protocol used in this benchmark to ensure a fair comparison.

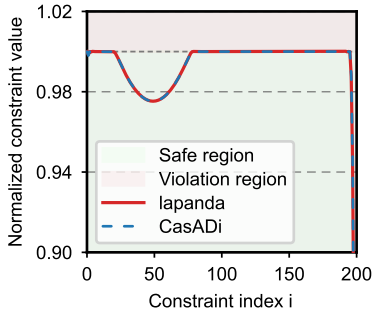


Figure 2: Normalized constraint values $\sqrt{x_i^2 + x_{i+1}^2}/r_i(\theta)$ for $n = 200$.

5.2 Imitation Learning with Constrained OCPs

We next evaluate `lapanda` on imitation learning tasks built from constrained OCPs. The goal is to learn problem parameters such that the MPC solution matches the target trajectory generated by a teacher controller. Accordingly, the imitation loss measures the discrepancy between the planned state-control trajectory and the demonstrated trajectory:

$$\mathcal{L}_{\text{imit}}(\theta) = \frac{1}{2N_{\text{demo}}} \sum_{j=1}^{N_{\text{demo}}} \left(\left\| X^\star(\theta;\xi_j) - X_j^{\text{demo}} \right\|_2^2 + \left\| U^\star(\theta;\xi_j) - U_j^{\text{demo}} \right\|_2^2 \right), \tag{16}$$

where N_{demo} is the number of demonstrations, and $X^\star(\theta;\xi_j), U^\star(\theta;\xi_j)$ denote the solution trajectory under parameter θ and initial condition ξ_j. We consider three representative constrained OCPs:

- **CartPole**: nonlinear underactuated dynamics with force bounds and a total control-energy constraint.
- **Quadrotor**: nonlinear flight dynamics with thrust and torque bounds, altitude and horizontal-position bounds, and attitude constraints.
- **Robot Arm**: linear joint-space dynamics with joint-velocity bounds and joint-position limits, together with nonlinear, nonconvex end-effector obstacle-avoidance geometry.

Open-loop learning. We first conduct open-loop imitation learning experiments. In this setting, each sample starts from an independent initial condition, and learning is driven by the discrepancy between the OCP solutions obtained with the current parameters and the teacher solutions generated using the target parameters. This setting evaluates whether the computed sensitivities support stable and efficient parameter learning across different initial conditions.

As shown in Fig. 3, we summarize the loss trends and average solve times on different constrained OCPs, comparing `lapanda` with `SafePDP`, an efficient PMP-based differentiable OCP solver, and `TurboMPC`, a recent GPU-parallel differentiable MPC solver. `TurboMPC` fails to converge on the CartPole task because it mainly supports pointwise inequalities, whereas this task includes a trajectory-level control-energy constraint. Across the tested tasks, `lapanda` achieves stable learning and the fastest solve time for these OCPs with multiple constraints. In comparison, the GPU implementation of `TurboMPC` is less effective at this problem scale, consistent with its reported constant-factor overhead from sparse direct linear solves.

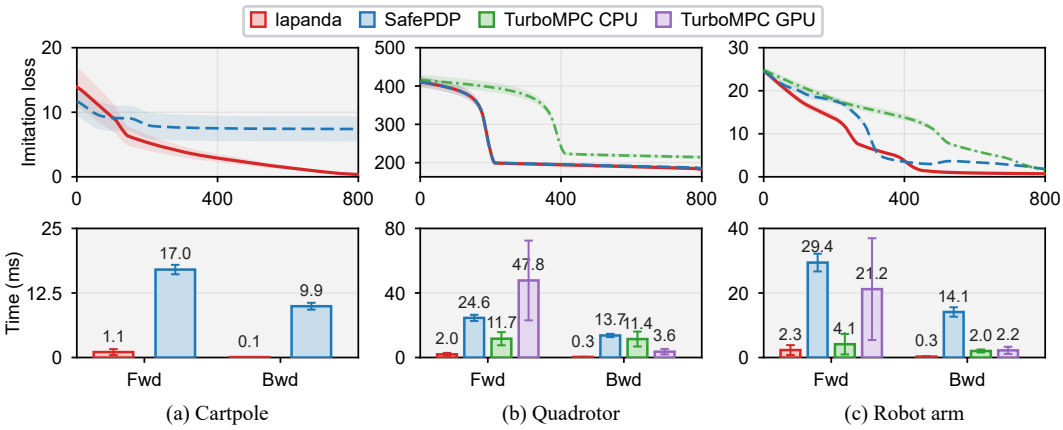


Figure 3: Open-loop imitation learning results on different constrained OCPs.

Closed-loop learning. We further evaluate closed-loop imitation learning. Given a demonstration trajectory generated by the target parameters, the learner repeatedly rolls out the system using the current parameters in a receding-horizon manner. Learning is driven by minimizing the discrepancy between the resulting closed-loop trajectory and the demonstration, thereby directly evaluating whether the learned parameters recover the desired closed-loop policy.

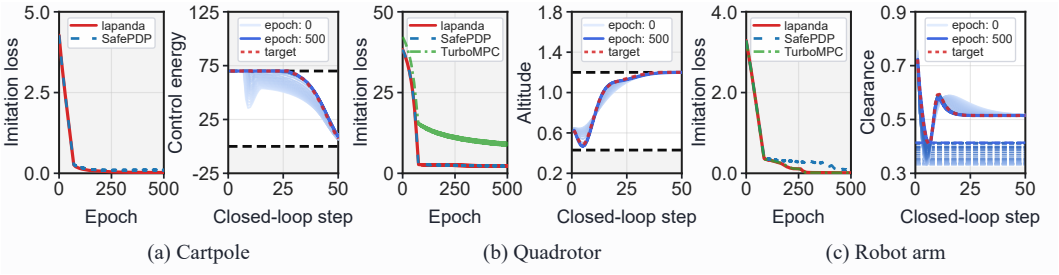


Figure 4: Closed-loop imitation learning results including training loss and policy evolution.

As shown in Fig. 4, `lapanda` achieves stable policy learning in the closed-loop setting. The constraint evolution shows that `lapanda` maintains the imposed constraints within the prescribed tolerance throughout training. In the Robot Arm task, the learned behavior also involves learning the constraint-related parameter, as reflected by the change in the obstacle-avoidance constraint during training. More detailed computational results are reported in Table 3.

Table 3: Detailed performance comparison of different solvers on closed-loop OCP benchmarks.

OCP	Method	Fwd. time (ms)	Bwd. time (ms)	Total time (ms)	Memory (MB)
CartPole	lapanda	**5.10**	**0.09**	**5.20**	**1.64**
	SafePDP	22.06	9.65	31.71	7.71
Quadrotor	lapanda	**1.20**	**0.24**	**1.45**	**1.79**
	SafePDP	29.86	13.41	43.27	9.23
	TurboMPC-CPU	10.46	7.74	18.20	241.30
	TurboMPC-GPU	21.78	2.28	24.06	336.48
Robot Arm	lapanda	**2.36**	**0.29**	**2.66**	**1.76**
	SafePDP	27.94	12.85	40.79	8.02
	TurboMPC-CPU	5.62	1.17	6.79	248.35
	TurboMPC-GPU	24.09	1.96	26.05	347.11

The solver runtimes exhibit a trend similar to that observed in the open-loop experiments. For memory usage, we report the sum of the solver construction overhead and the peak temporary memory overhead during the solve. `TurboMPC` incurs a heavier memory footprint due to its GPU and parallel implementation, whereas `lapanda` remains substantially more lightweight. Detailed problem formulations and additional experimental results are provided in Appendix C.2, and all solvers in the above experiments use the same stopping tolerance of 10^{-3}.

5.3 Embedded Deployment

Finally, we evaluate `lapanda` on an embedded mobile robot equipped with a six-core Arm® Cortex®-A78AE processor. We consider two obstacle-avoidance tasks: circular obstacle avoidance with smooth nonlinear constraints and rectangular obstacle avoidance formulated as an MPCC. We compare `lapanda` with `acados`, a representative embedded nonlinear optimal control solver, and summarize the results in Table 4.

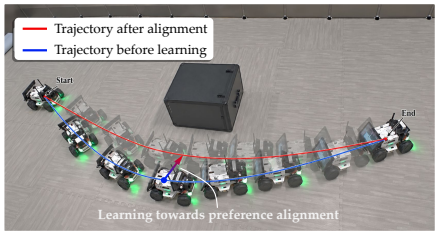
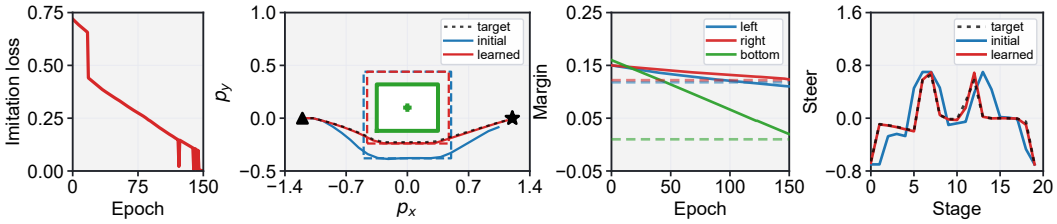


Figure 5: Mobile robot trajectory alignment.

Table 4: Embedded deployment performance comparison.

Obstacle	Method	Success	Fwd. (ms)	Bwd. (ms)	Memory (MB)
Circle	lapanda	✓	1.76	0.16	2.16
	acados	✓	3.27	0.24	4.52
Rectangle	lapanda	✓	55.5	1.13	2.60
	acados	✗	–	–	–

As summarized in Table 4, both `lapanda` and `acados` successfully solve the circular obstacle-avoidance task. The rectangular task is more challenging because its MPCC formulation violates LICQ and thereby induces a singular KKT matrix. Consequently, `acados`, which relies on local KKT models, fails to converge under the tested configurations. In contrast, the first-order `PANDA` inner solver of `lapanda` relies on a proximal-residual optimality measure rather than KKT systems. Combined with ALM constraint penalization, it can progressively approach the feasible set from infeasible iterates, empirically converging outside our theoretical assumptions and supporting successful imitation learning. The learning process is shown in Fig. 6. We further evaluate both solvers using a smooth approximation of the rectangular-obstacle constraint in Appendix C.3, where the detailed problem formulations and solver configurations are provided.



(a) Loss (b) Trajectory (c) Margins (d) Steer

Figure 6: Learning performance for the rectangular obstacle-avoidance imitation task.

6 Conclusion

We proposed `lapanda`, a matrix-free differentiable solver for nonconvex optimization layers with general constraints. We established its optimization and sensitivity principles and validated its effectiveness through extensive experiments, demonstrating fast computation, low memory consumption, reliable constraint satisfaction, and stable learning performance. With efficient implementations and user-friendly interfaces across multiple platforms (Appendix D), `lapanda` provides a practical foundation for extending differentiable optimization to broader learning and control applications.

Limitations and future work. First, the finite ALM penalties used in `lapanda` introduce approximation errors in the backward sensitivity. Although stable learning is observed across most experiments, further theoretical characterization and matrix-free differentiation of the original KKT system warrant investigation. Second, `lapanda` requires careful tuning of multiplier initialization, penalty updates, and `PANDA` parameters. Adaptive parameter strategies could therefore reduce manual tuning and improve robustness. Third, as a differentiable solver for nonconvex optimization layers with general constraints, `lapanda` does not yet exploit the temporal sparsity and dynamical structure specific to OCPs, leaving room for further scalability improvements. Finally, the current implementation targets lightweight CPU execution. Extending `lapanda` to GPU-parallel and batched computation represents a natural next step.

AI USE STATEMENT

Generative AI tools were used solely to assist with language polishing, including edits for grammar, clarity, concision, and readability. All AI-assisted edits were reviewed and validated by the authors, who take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The assumptions and complete proofs supporting the theoretical results are provided in Appendix B. Appendix C reports the problem formulations, parameter values, solver settings, evaluation metrics, and timing and memory measurement protocols for the experiments. Appendix D describes the implementation and user interfaces. The anonymous code repository linked in the abstract provides the source code, configuration files, and scripts needed to reproduce the reported results.

REFERENCES

Flavia Sofia Acerbo, Jan Swevers, Tinne Tuytelaars, and Tong Duy Son. Evaluation of MPC-based imitation learning for human-like autonomous driving. *IFAC-PapersOnLine*, 56(2):4871–4876, 2023.

Flavia Sofia Acerbo, Jan Swevers, Tinne Tuytelaars, and Tong Duy Son. Driving from vision through differentiable optimal control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2824–2831. IEEE, 2024.

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, 2019a.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019b. URL https://arxiv.org/pdf/1904.09043.

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

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

Joel AE Andersson and James B Rawlings. Sensitivity analysis for nonlinear programming in CasADi. *IFAC-PapersOnLine*, 51(20):331–336, 2018.

Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi: A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.

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, pp. 5230–5242, 2022.

Gabriel Bravo-Palacios, Jianghan Zhang, Zachary Pestrikov, Brian Plancher, and Thomas Lew. TurboMPC: Fast, scalable, and differentiable model predictive control on the GPU. *arXiv preprint arXiv:2606.24039*, 2026. URL https://arxiv.org/abs/2606.24039.

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

Yuankun Chen, Zifei Nie, Xun Gong, Yunfeng Hu, and Hong Chen. PANDA: A matrix-free differentiable NMPC solver via proximal averaged Quasi-Newton with adaptive linesearch algorithm. *arXiv preprint arXiv:2608.16280*, 2026. URL https://arxiv.org/abs/2608.16280.

Jonathan Frey, Katrin Baumgärtner, Gianluca Frison, Dirk Reinhardt, Jasper Hoffmann, Leonard Fichtner, Sebastien Gros, and Moritz Diehl. Differentiable nonlinear model predictive control. *arXiv preprint arXiv:2505.01353*, 2025.

Sébastien Gros and Moritz Diehl. Numerical optimal control, 2024. URL https://www.syscop.de/files/misc/book-NOCSE.pdf.

Zhiyu Huang, Peter Karkus, Boris Ivanovic, Yuxiao Chen, Marco Pavone, and Chen Lv. DTPP: Differentiable joint conditional prediction and cost evaluation for tree policy planning in autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6806–6812. IEEE, 2024a.

Zhiyu Huang, Haochen Liu, Jingda Wu, and Chen Lv. Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving. *IEEE transactions on neural networks and learning systems*, 35(11):15222–15236, 2024b.

Wanxin Jin, Shaoshuai Mou, and George J Pappas. Safe pontryagin differentiable programming. *Advances in Neural Information Processing Systems*, 34:16034–16050, 2021.

Peter Karkus, Boris Ivanovic, Shie Mannor, and Marco Pavone. DiffStack: A differentiable and modular control stack for autonomous vehicles. In *Conference on robot learning*, pp. 2170–2180. PMLR, 2023.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021.

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

Pieter Pas, Mathijs Schuurmans, and Panagiotis Patrinos. Alpaqa: A matrix-free solver for nonlinear MPC and large-scale nonconvex optimization. In *2022 European Control Conference (ECC)*, pp. 417–422. IEEE, 2022.

Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky TQ Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, et al. Theseus: A library for differentiable nonlinear optimization. *Advances in Neural Information Processing Systems*, 35:3801–3818, 2022.

Angel Romero, Yunlong Song, and Davide Scaramuzza. Actor-critic model predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14777–14784. IEEE, 2024.

Lorenzo Stella, Andreas Themelis, Pantelis Sopasakis, and Panagiotis Patrinos. A simple and efficient algorithm for nonlinear model predictive control. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 1939–1944. IEEE, 2017.

Andreas Themelis, Lorenzo Stella, and Panagiotis Patrinos. Forward-backward envelope for the sum of two nonconvex functions: Further properties and nonmonotone linesearch algorithms. *SIAM Journal on Optimization*, 28(3):2274–2303, 2018.

Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados—a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14(1):147–183, 2022.

Weikang Wan, Ziyu Wang, Yufei Wang, Zackory Erickson, and David Held. DiffTORI: Differentiable trajectory optimization for deep reinforcement and imitation learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 109430–109459, 2024.

Zhengtong Xu and Yu She. LeTO: Learning constrained visuomotor policy with differentiable trajectory optimization. *IEEE Transactions on Automation Science and Engineering*, 22:8567–8578, 2025.

Zihao Zhao, Kai-Chia Mo, Shing-Hei Ho, Brandon Amos, and Kai Wang. A fully first-order layer for differentiable optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.02494.

APPENDIX

A PROBLEM DEFINITION AND OCP TRANSCRIPTION

This appendix presents the single-shooting transcription adopted for constrained optimal control problems. Consider the finite-horizon problem

$$\begin{aligned} \min_{\{x_k,u_k\}} \quad & \sum_{k=0}^{N-1} \ell_k(x_k,u_k;\theta) + \ell_N(x_N;\theta) \\ \text{s.t.} \quad & x_0 = x_{\text{init}}(\theta), \\ & x_{k+1} = f_k(x_k,u_k;\theta), \\ & c_k(x_k,u_k;\theta) = 0, \quad h_k(x_k,u_k;\theta) \le 0, \\ & c_N(x_N;\theta) = 0, \quad h_N(x_N;\theta) \le 0, \\ & u_k \in \mathbf{U}_k, \quad k = 0,\dots,N-1. \end{aligned} \tag{17}$$

In single shooting, the state variables are eliminated by recursively applying the system dynamics. For the stacked control vector $u := (u_0^\top, \dots, u_{N-1}^\top)^\top$, the state rollout is defined as

$$x_0(u;\theta) = x_{\text{init}}(\theta), \qquad x_{k+1}(u;\theta) = f_k(x_k(u;\theta),u_k;\theta). \tag{18}$$

Substituting Eq. (18) into the objective and constraints gives

$$\ell(u,\theta) := \sum_{k=0}^{N-1} \ell_k(x_k(u;\theta),u_k;\theta) + \ell_N(x_N(u;\theta);\theta), \tag{19a}$$

and

$$c(u,\theta) := \begin{bmatrix} c_0(x_0(u;\theta),u_0;\theta) \\ \vdots \\ c_{N-1}(x_{N-1}(u;\theta),u_{N-1};\theta) \\ c_N(x_N(u;\theta);\theta) \end{bmatrix}, \quad h(u,\theta) := \begin{bmatrix} h_0(x_0(u;\theta),u_0;\theta) \\ \vdots \\ h_{N-1}(x_{N-1}(u;\theta),u_{N-1};\theta) \\ h_N(x_N(u;\theta);\theta) \end{bmatrix}. \tag{19b}$$

The simple constraint set becomes $\mathbf{U} := \mathbf{U}_0 \times \cdots \times \mathbf{U}_{N-1}$. Therefore, Eq. (17) is reformulated as the control-only constrained optimization problem

$$\min_{u\in\mathbf{U}} \ \ell(u,\theta) \quad \text{s.t.} \quad c(u,\theta) = 0, \qquad h(u,\theta) \le 0, \tag{20}$$

which is the form of Problem (P).

B THEORETICAL DETAILS

B.1 ELIMINATING THE SLACK VARIABLE

For fixed u, the z-subproblem in Eq. (4) is

$$\min_{z\in\mathcal{D}} \lambda^\top (G(u,\theta) - z) + \frac{\rho}{2}\|G(u,\theta) - z\|^2. \tag{21}$$

Completing the square gives

$$\frac{\rho}{2}\left\|G(u,\theta) + \rho^{-1}\lambda - z\right\|^2 - \frac{1}{2\rho}\|\lambda\|^2, \tag{22}$$

so that

$$z^\star(u) = \Pi_{\mathcal{D}}\left(G(u,\theta) + \rho^{-1}\lambda\right). \tag{23}$$

Substituting this minimizer into the augmented Lagrangian and dropping the constant term $-\|\lambda\|^2/(2\rho)$ yields the reduced objective

$$\psi_\rho(u;\theta,\lambda) = \ell(u,\theta) + \frac{\rho}{2}\,\mathrm{dist}^2\left(G(u,\theta) + \rho^{-1}\lambda, \mathcal{D}\right),$$

which gives Eq. (5) and Eq. (6).

B.2 PANDA INNER SOLVER

Given the ALM variables (λ_r, ρ_r), the reduced inner subproblem takes the composite form

$$\min_u \ \varphi_r(u) := \psi_{\rho_r}(u; \theta, \lambda_r) + \delta_{\mathbf{U}}(u), \tag{24}$$

where ψ_{ρ_r} is the smooth reduced ALM objective and $\delta_{\mathbf{U}}$ encodes the simple constraint on u. For a stepsize $\gamma > 0$, define the forward-backward mapping

$$T_{\gamma,r}(u) := \operatorname{prox}_{\gamma\delta_{\mathbf{U}}}\left(u - \gamma\nabla_u\psi_{\rho_r}(u; \theta, \lambda_r)\right) = \Pi_{\mathbf{U}}\left(u - \gamma\nabla_u\psi_{\rho_r}(u; \theta, \lambda_r)\right), \tag{25}$$

and the associated forward-backward residual

$$R_{\gamma,r}(u) := \frac{1}{\gamma}\left(u - T_{\gamma,r}(u)\right). \tag{26}$$

A point satisfying $R_{\gamma,r}(u) = 0$ is an R-critical point. For the indicator term $\delta_{\mathbf{U}}$, this condition is equivalent to

$$0 \in \nabla_u\psi_{\rho_r}(u; \theta, \lambda_r) + N_{\mathbf{U}}(u), \tag{27}$$

and hence it characterizes a stationary point of the reduced ALM subproblem.

PANDA solves the residual equation $R_{\gamma,r}(u) = 0$ by combining three mechanisms. (i) It uses the forward-backward mapping as a descent step for the composite objective and uses the residual norm as the stationarity measure. (ii) It constructs a limited-memory quasi-Newton correction for the residual equation, so that the local convergence is faster than a plain projected gradient iteration. (iii) It globalizes the quasi-Newton correction through an averaged trial point and an FBE-based line search. In addition, PANDA adapts the stepsize γ: it reduces γ when the local upper model is not valid, and attempts to enlarge γ when the current stepsize becomes too conservative.

The simplified inner routine is summarized in Algorithm 2. The output is an approximate R-critical point of Eq. (24), which is later used for implicit differentiation through $R_{\gamma,r}(u_{r+1}) = 0$.

Algorithm 2 PANDA inner solver for the reduced ALM subproblem

Input: u^0, $\gamma_0 > 0$, tolerance ε_{in}, ALM variables (λ_r, ρ_r)
for $k = 0, 1, \ldots$ **do**
  Compute $\bar{u}^k = T_{\gamma_k,r}(u^k)$ and $R_{\gamma_k,r}(u^k) = \gamma_k^{-1}(u^k - \bar{u}^k)$.
  if $\|R_{\gamma_k,r}(u^k)\| \le \varepsilon_{\text{in}}$ **then**
    return u^k
  end if
  Construct a quasi-Newton direction $d^k \approx -H_k R_{\gamma_k,r}(u^k)$.
  Form the averaged trial point $u^k_{\text{trial}} = (1 - \sigma_k)\bar{u}^k + \sigma_k(u^k + d^k)$, with $\sigma_k \in (0, 1]$.
  Backtrack σ_k until the FBE sufficient decrease condition holds.
  Adapt γ_k according to the local upper-model condition.
  Set $u^{k+1} = u^k_{\text{trial}}$.
end for

B.3 PROOF OF THEOREM 1

We prove the three statements in Theorem 1. The argument follows the local method-of-multipliers proof in Bertsekas (1999, Proposition 4.2.3) and the positive-definiteness argument in Nocedal & Wright (2006, Theorem 17.5), with the difference that we apply the argument to the slack reformulation and retain the simple constraints $\mathbf{U} \times \mathcal{D}$.

Set $y = (u, z)$, $\mathcal{C} = \mathbf{U} \times \mathcal{D}$, and $r(y, \theta) = G(u, \theta) - z$. Then the slack problem is locally

$$\min_{y \in \mathcal{C}} \ \ell(u, \theta) \quad \text{s.t.} \quad r(y, \theta) = 0. \tag{28}$$

By strict complementarity, the active constraints of $\mathcal{C}$ are locally identified. We denote the corresponding active manifold by $b(y) = 0$, with multiplier ν, and define

$$\mathscr{L}(y, \lambda, \nu; \theta) = \ell(u, \theta) + \langle \lambda, r(y, \theta)\rangle + \langle \nu, b(y)\rangle. \tag{29}$$

Let $y^\star = (u^\star, z^\star)$, $J_r^\star = \nabla_y r(y^\star, \theta)$, $J_b^\star = \nabla b(y^\star)$, and $W^\star = \nabla^2_{yy}\mathscr{L}(y^\star, \lambda^\star, \nu^\star; \theta)$. LICQ means that $\begin{bmatrix} J_r^\star \\ J_b^\star \end{bmatrix}$ has full row rank, and SOSC means that $d^\top W^\star d > 0$ for all nonzero d satisfying $J_r^\star d = 0$ and $J_b^\star d = 0$.

Claim (i). For fixed (λ_r, ρ_r), the slack ALM subproblem is

$$\min_{y \in \mathcal{C}} \quad \ell(u,\theta) + \lambda_r^\top r(y,\theta) + \frac{\rho_r}{2}\|r(y,\theta)\|^2, \tag{30}$$

The local KKT conditions of Eq. (30) are

$$\begin{aligned} 0 &= \nabla_y \ell(u,\theta) + \nabla_y r(y,\theta)^\top (\lambda_r + \rho_r r(y,\theta)) + \nabla b(y)^\top \nu, \\ 0 &= b(y). \end{aligned} \tag{31}$$

Define the shifted multiplier $\lambda := \lambda_r + \rho_r r(y,\theta)$. Then Eq. (31), together with the multiplier relation, can be written as

$$\begin{aligned} 0 &= \nabla_y \ell(u,\theta) + \nabla_y r(y,\theta)^\top \lambda + \nabla b(y)^\top \nu, \\ 0 &= r(y,\theta) - \rho_r^{-1}(\lambda - \lambda_r), \\ 0 &= b(y). \end{aligned} \tag{32}$$

We first verify the second-order condition for the ALM subproblem. At $y^\star$, since $r(y^\star,\theta) = 0$, the Hessian of the ALM objective on the identified active manifold is $W^\star + \rho J_r^{\star\top} J_r^\star$, where $W^\star = \nabla^2_{yy}\mathscr{L}(y^\star,\lambda^\star,\nu^\star;\theta)$. By LICQ and SOSC, there exists $\bar\rho > 0$ such that

$$d^\top \left(W^\star + \rho J_r^{\star\top} J_r^\star\right) d > 0 \quad \forall d \neq 0, \quad J_b^\star d = 0, \quad \rho \geq \bar\rho. \tag{33}$$

Indeed, if this were false, then there would exist $\rho_k \to \infty$ and $\|d_k\| = 1$, with $J_b^\star d_k = 0$, such that

$$d_k^\top W^\star d_k + \rho_k \|J_r^\star d_k\|^2 \leq 0. \tag{34}$$

Hence $J_r^\star d_k \to 0$. Taking a convergent subsequence gives a nonzero $\bar d$ satisfying $J_b^\star \bar d = 0$ and $J_r^\star \bar d = 0$, which contradicts SOSC.

Set $t := \rho_r^{-1}(\lambda_r - \lambda^\star)$ and $\kappa := \rho_r^{-1}$. Then Eq. (32) is equivalently written as

$$\Phi(y,\lambda,\nu;t,\kappa) = 0, \tag{35}$$

where

$$\Phi(y,\lambda,\nu;t,\kappa) = \begin{bmatrix} \nabla_y \ell(u,\theta) + \nabla_y r(y,\theta)^\top \lambda + \nabla b(y)^\top \nu \\ r(y,\theta) + t + \kappa\lambda^\star - \kappa\lambda \\ b(y) \end{bmatrix}.$$

For $t = 0$, the point $(y^\star,\lambda^\star,\nu^\star)$ solves Eq. (35) for all $\kappa \in [0, \bar\rho^{-1}]$. The Jacobian of Φ with respect to (y,λ,ν) at this point is

$$K_\kappa = \begin{bmatrix} W^\star & J_r^{\star\top} & J_b^{\star\top} \\ J_r^\star & -\kappa I & 0 \\ J_b^\star & 0 & 0 \end{bmatrix}. \tag{36}$$

We claim that K_κ is nonsingular for all $\kappa \in [0, \bar\rho^{-1}]$. For $\kappa = 0$, this follows from the standard KKT matrix nonsingularity under LICQ and SOSC. For $\kappa > 0$, suppose $K_\kappa(d,p,s) = 0$. Then

$$\begin{aligned} W^\star d + J_r^{\star\top} p + J_b^{\star\top} s &= 0, \\ J_r^\star d - \kappa p &= 0, \\ J_b^\star d &= 0. \end{aligned} \tag{37}$$

Multiplying the first equation in Eq. (37) by $d^\top$ and using the second and third equations gives

$$d^\top W^\star d + \kappa^{-1}\|J_r^\star d\|^2 = 0. \tag{38}$$

By Eq. (33), this implies $d = 0$. Then $J_r^{\star\top} p + J_b^{\star\top} s = 0$, and LICQ gives $p = 0$ and $s = 0$. Thus K_κ is nonsingular.

Since K_κ depends continuously on κ and is nonsingular on the compact interval $[0, \bar\rho^{-1}]$, its inverse is uniformly bounded. The parameterized implicit function theorem therefore provides a common local neighborhood for all $\kappa \in [0, \bar\rho^{-1}]$. Consequently, there exist $\delta > 0$ and a neighborhood of $(y^\star,\lambda^\star,\nu^\star)$ such that, whenever $\rho_r \geq \bar\rho$ and $\|\lambda_r - \lambda^\star\| \leq \rho_r\delta$, the system in Eq. (35) has a unique local solution $(y_{r+1},\lambda_{r+1},\nu_{r+1})$. Equivalently, $y_{r+1} = (u_{r+1}, z_{r+1})$ is the unique local solution of the slack ALM subproblem in Eq. (30), and $\lambda_{r+1} = \lambda_r + \rho_r r(y_{r+1},\theta)$.

Finally, by taking the neighborhood smaller if necessary, strict complementarity keeps the active set of $\mathcal{C}$ fixed, LICQ persists by continuity of the active constraint gradients, and the positive-definiteness condition in Eq. (33) gives SOSC at y_{r+1}. This proves claim (i).

Claim (ii). From the implicit function construction in claim (i), there exist local C^1 functions $y(t,\kappa)$ and $\lambda(t,\kappa)$ such that $y_{r+1} = y(t_r, \kappa_r)$ and $\lambda_{r+1} = \lambda(t_r, \kappa_r)$, where $t_r = \rho_r^{-1}(\lambda_r - \lambda^\star)$ and $\kappa_r = \rho_r^{-1}$. Moreover, $y(0,\kappa) = y^\star$ and $\lambda(0,\kappa) = \lambda^\star$ for all $\kappa \in [0, \bar{\rho}^{-1}]$.

Since the implicit functions are C^1, and since $\kappa \in [0, \bar{\rho}^{-1}]$ belongs to a compact interval, their derivatives with respect to t are uniformly bounded. Hence, there exists $M > 0$ such that, for all sufficiently small t and all $\kappa \in [0, \bar{\rho}^{-1}]$,

$$\|y(t,\kappa) - y(0,\kappa)\| + \|\lambda(t,\kappa) - \lambda(0,\kappa)\| \le M\|t\|. \tag{39}$$

Taking $t = t_r$ and $\kappa = \kappa_r$ in Eq. (39) gives

$$\|y_{r+1} - y^\star\| + \|\lambda_{r+1} - \lambda^\star\| \le M\|t_r\|. \tag{40}$$

Using $t_r = \rho_r^{-1}(\lambda_r - \lambda^\star)$, we obtain

$$\|y_{r+1} - y^\star\| + \|\lambda_{r+1} - \lambda^\star\| \le M\frac{\|\lambda_r - \lambda^\star\|}{\rho_r}. \tag{41}$$

Since $y = (u, z)$, Eq. (41) directly implies the estimates in claim (ii).

Claim (iii). By Appendix B.1, eliminating z gives

$$z_{\rho_r}(u) = \Pi_{\mathcal{D}}\left(G(u,\theta) + \rho_r^{-1}\lambda_r\right),$$

and the reduced composite subproblem

$$\min_{u\in\mathbf{U}} \ \psi_{\rho_r}(u;\theta,\lambda_r) + \delta_{\mathbf{U}}(u).$$

Since the slack ALM subproblem satisfies LICQ and SOSC at y_{r+1}, it has local quadratic growth on the identified active manifold: for nearby feasible $y = (u, z)$,

$$\mathcal{L}_{\rho_r}(y,\lambda_r;\theta) \ge \mathcal{L}_{\rho_r}(y_{r+1},\lambda_r;\theta) + \alpha\|y - y_{r+1}\|^2 \tag{42}$$

for some $\alpha > 0$. Taking $z = z_{\rho_r}(u)$ and using $z_{r+1} = z_{\rho_r}(u_{r+1})$, we obtain

$$\psi_{\rho_r}(u;\theta,\lambda_r) \ge \psi_{\rho_r}(u_{r+1};\theta,\lambda_r) + \alpha\|u - u_{r+1}\|^2 \tag{43}$$

for all nearby $u \in \mathbf{U}$. Thus u_{r+1} is a strong local minimizer of the reduced composite subproblem.

It remains to justify prox-regularity. By Assumption 1, $\mathbf{U}$ admits a local C^2 active-manifold representation $b_{\mathbf{U}}(u) = 0$, and LICQ holds on this manifold. Hence every nearby normal vector $v \in N_{\mathbf{U}}(u)$ admits the representation $v = \nabla b_{\mathbf{U}}(u)^\top \xi$, with $\|\xi\| \le c_0\|v\|$. For nearby $u, \bar{u} \in \mathbf{U}$, Taylor expansion of $b_{\mathbf{U}}(\bar{u}) = b_{\mathbf{U}}(u) = 0$ gives

$$\left\langle \nabla b_{\mathbf{U}}(u)^\top \xi, \bar{u} - u \right\rangle \le c_1\|\xi\|\,\|\bar{u} - u\|^2. \tag{44}$$

Therefore,

$$\langle v, \bar{u} - u\rangle \le c_0 c_1 \|v\|\,\|\bar{u} - u\|^2, \tag{45}$$

which is the local prox-regularity inequality. Thus $\mathbf{U}$ is prox-regular around u_{r+1}, equivalently $\delta_{\mathbf{U}}$ is locally prox-regular relative to the identified active manifold.

Together with the local smoothness of ψ_{ρ_r}, these properties satisfy the local regularity assumptions of the PANDA convergence result. Hence, the PANDA inner iteration is locally well posed and converges locally to u_{r+1}. This proves claim (iii), and the proof is complete.

B.4 Sensitivity of the Final lapanda Subproblem

Recall that the reduced subproblem at the r-th ALM iteration is

$$u_{r+1}(\theta) \in \operatorname*{argmin}_{u\in\mathbf{U}} \psi_{\rho_r}(u;\theta,\lambda_r), \tag{46}$$

where $\psi_{\rho_r}(u;\theta,\lambda_r) = \ell(u,\theta) + \frac{\rho_r}{2}\operatorname{dist}^2\left(G(u,\theta) + \rho_r^{-1}\lambda_r, \mathcal{D}\right)$.

As in Eq. (9), let $a(u,\theta) = 0$ collect the equality and locally active inequality constraints, and let $\phi(u) = 0$ describe the identified active manifold of $\mathbf{U}$. At u_{r+1}, define the shifted active-set

multiplier $\bar{\eta}_{r+1} := \eta_r + \rho_r a(u_{r+1}, \theta)$. Here, η_r contains the corresponding components of λ_r. The inactive inequality components do not enter the local reduced system because their projections remain in the interior of $\mathbb{R}_-$. Let $A_{r+1} := \nabla_u a(u_{r+1}, \theta)$ and $C_{r+1} := \nabla_u \phi(u_{r+1})$, and let μ_{r+1} be the multiplier of $\phi(u) = 0$. The local KKT conditions are

$$\begin{aligned} \nabla_u \ell(u_{r+1}, \theta) + A_{r+1}^\top \bar{\eta}_{r+1} + C_{r+1}^\top \mu_{r+1} = 0, \\ a(u_{r+1}, \theta) - \rho_r^{-1} (\bar{\eta}_{r+1} - \eta_r) = 0, \\ \phi(u_{r+1}) = 0. \end{aligned} \tag{47}$$

The second condition is the definition of $\bar{\eta}_{r+1}$ rearranged. Throughout the differentiation, η_r and ρ_r are treated as fixed. Differentiating the stationarity condition with respect to θ gives

$$H_{r+1} D_\theta u_{r+1} + A_{r+1}^\top D_\theta \bar{\eta}_{r+1} + C_{r+1}^\top D_\theta \mu_{r+1} + B_{r+1} = 0, \tag{48}$$

where $H_{r+1} := \nabla_{uu}^2 \left(\ell + \bar{\eta}_{r+1}^\top a + \mu_{r+1}^\top \phi\right)$, $B_{r+1} := \nabla_{u\theta}^2 \left(\ell + \bar{\eta}_{r+1}^\top a + \mu_{r+1}^\top \phi\right)$, with all derivatives evaluated at $(u_{r+1}, \theta, \bar{\eta}_{r+1}, \mu_{r+1})$. Differentiating the shifted active-constraint relation gives

$$A_{r+1} D_\theta u_{r+1} - \rho_r^{-1} D_\theta \bar{\eta}_{r+1} + D_{r+1} = 0, \tag{49}$$

where $D_{r+1} := \nabla_\theta a(u_{r+1}, \theta)$.

Finally, since $\mathbf{U}$ is independent of θ, differentiating $\phi(u_{r+1}) = 0$ gives

$$C_{r+1} D_\theta u_{r+1} = 0. \tag{50}$$

Stacking Eqs. (48)–(50) yields

$$\begin{bmatrix} H_{r+1} & A_{r+1}^\top & C_{r+1}^\top \\ A_{r+1} & -\rho_r^{-1} I & 0 \\ C_{r+1} & 0 & 0 \end{bmatrix} \begin{bmatrix} D_\theta u_{r+1} \\ D_\theta \bar{\eta}_{r+1} \\ D_\theta \mu_{r+1} \end{bmatrix} = - \begin{bmatrix} B_{r+1} \\ D_{r+1} \\ 0 \end{bmatrix}. \tag{51}$$

B.5 Proof of Theorem 2

We prove Theorem 2. Let

$$d^\star := \begin{bmatrix} D_\theta u^\star \\ D_\theta \eta^\star \\ D_\theta \mu^\star \end{bmatrix}, \qquad d_{r+1} := \begin{bmatrix} D_\theta u_{r+1} \\ D_\theta \bar{\eta}_{r+1} \\ D_\theta \mu_{r+1} \end{bmatrix}. \tag{52}$$

The original active-set KKT sensitivity system and the reduced `lapanda` sensitivity system can be written compactly as

$$K d^\star = -\beta, \qquad K_{r+1} d_{r+1} = -\beta_{r+1}, \tag{53}$$

where

$$K = \begin{bmatrix} H & A^\top & C^\top \\ A & 0 & 0 \\ C & 0 & 0 \end{bmatrix}, \qquad K_{r+1} = \begin{bmatrix} H_{r+1} & A_{r+1}^\top & C_{r+1}^\top \\ A_{r+1} & -\rho_r^{-1} I & 0 \\ C_{r+1} & 0 & 0 \end{bmatrix},$$

and

$$\beta = \begin{bmatrix} B \\ D \\ 0 \end{bmatrix}, \qquad \beta_{r+1} = \begin{bmatrix} B_{r+1} \\ D_{r+1} \\ 0 \end{bmatrix}.$$

The third block is zero because the active manifold $\phi(u) = 0$ of the simple set $\mathbf{U}$ is independent of θ. If $\mathbf{U}$ depends on θ, the corresponding blocks are replaced by E and E_{r+1}.

Under Assumption 1, strict complementarity implies local active-set identification. Hence, for all sufficiently advanced ALM iterates, the active manifolds used in Eqs. (11) and (51) coincide locally. Moreover, LICQ and SOSC imply that the original active-set KKT matrix K is nonsingular.

We next compare the two linear systems. By Theorem 1, the evaluated quantities $(u_{r+1}, \bar{\eta}_{r+1}, \mu_{r+1})$ remain in a neighborhood of $(u^\star, \eta^\star, \mu^\star)$. Since the active functions are $C^{2,1}$, the Jacobian and Lagrangian-Hessian blocks are locally Lipschitz in the primal-dual variables. Therefore,

$$H_{r+1} - H = O(\epsilon_{r+1}), \qquad A_{r+1} - A = O(\epsilon_{r+1}), \qquad C_{r+1} - C = O(\epsilon_{r+1}), \tag{54}$$

where $\epsilon_{r+1} := \|u_{r+1} - u^\star\| + \|\bar{\eta}_{r+1} - \eta^\star\| + \|\mu_{r+1} - \mu^\star\|$.

The finite-penalty block is the only structural difference between the two systems. Therefore,

$$K_{r+1} = K + \Delta_{\rho_r} + \mathcal{E}_{r+1}, \qquad \Delta_{\rho_r} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\rho_r^{-1} I & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \|\mathcal{E}_{r+1}\| = O(\epsilon_{r+1}). \tag{55}$$

Consequently,

$$\|K_{r+1} - K\| = O(\rho_r^{-1}) + O(\epsilon_{r+1}). \tag{56a}$$

The same $C^{2,1}$ regularity yields

$$\|\beta_{r+1} - \beta\| = O(\epsilon_{r+1}). \tag{56b}$$

Since K is nonsingular, the perturbation estimate in Eq. (56a) implies that K_{r+1} is nonsingular whenever ρ_r is sufficiently large and ϵ_{r+1} is sufficiently small. Moreover, its inverse is locally uniformly bounded: there exists $\kappa_K > 0$ such that

$$\|K_{r+1}^{-1}\| \le \kappa_K. \tag{56c}$$

Subtracting the two systems in Eq. (53) gives

$$K_{r+1}(d_{r+1} - d^\star) = -(\beta_{r+1} - \beta) - (K_{r+1} - K)d^\star.$$

Hence,

$$d_{r+1} - d^\star = -K_{r+1}^{-1}(\beta_{r+1} - \beta) - K_{r+1}^{-1}(K_{r+1} - K)d^\star.$$

Using Eqs. (56a), (56b), and (56c), we obtain

$$\|d_{r+1} - d^\star\| \le \kappa_K \left(\|\beta_{r+1} - \beta\| + \|K_{r+1} - K\|\,\|d^\star\|\right) = O(\rho_r^{-1}) + O(\epsilon_{r+1}). \tag{57}$$

Taking the first block yields

$$D_\theta u_{r+1} = D_\theta u^\star + O(\rho_r^{-1}) + O(\epsilon_{r+1}). \tag{58}$$

It remains to show that $\epsilon_{r+1} \to 0$ as the ALM outer iteration converges. The local ALM convergence result in Theorem 1 gives

$$u_{r+1} \to u^\star, \qquad \bar{\eta}_{r+1} \to \eta^\star. \tag{59}$$

The multiplier μ_{r+1} is associated with the identified active manifold of $\mathbf{U}$. The local stationarity condition of the final reduced subproblem is

$$\nabla_u \ell(u_{r+1}, \theta) + A_{r+1}^\top \bar{\eta}_{r+1} + C_{r+1}^\top \mu_{r+1} = 0. \tag{60}$$

By LICQ, the active-set multiplier is locally unique and bounded. Hence, $\{\mu_{r+1}\}$ is bounded and admits convergent subsequences. Let $\bar{\mu}$ be any accumulation point. Passing to the limit in Eq. (60) gives

$$\nabla_u \ell(u^\star, \theta) + A^\top \eta^\star + C^\top \bar{\mu} = 0. \tag{61}$$

The original active-set KKT stationarity condition is

$$\nabla_u \ell(u^\star, \theta) + A^\top \eta^\star + C^\top \mu^\star = 0. \tag{62}$$

Subtracting Eq. (62) from Eq. (61) gives $C^\top(\bar{\mu} - \mu^\star) = 0$. Since the active constraint gradients are linearly independent, C has full row rank, and therefore $\bar{\mu} = \mu^\star$. Thus every accumulation point equals $\mu^\star$, which implies

$$\mu_{r+1} \to \mu^\star \qquad \text{and} \qquad \epsilon_{r+1} \to 0. \tag{63}$$

Therefore, as the ALM outer iteration converges, the $O(\epsilon_{r+1})$ term in Eq. (58) vanishes. If the PANDA inner subproblems are solved inexactly, the same conclusion holds provided that their stationarity residuals converge to zero. If, in addition, $\rho_r \to \infty$, then the finite-penalty term $O(\rho_r^{-1})$ also vanishes, and hence

$$D_\theta u_{r+1} \longrightarrow D_\theta u^\star. \tag{64}$$

B.6 MATRIX-FREE OPERATORS

We derive the matrix-free backward operators for the final reduced `lapanda` subproblem from the residual equation

$$R_{\gamma,r}(u,\theta) = \frac{1}{\gamma}\left[u - \Pi_{\mathbf{U}}\left(u - \gamma\nabla_u \psi_{\rho_r}(u;\theta,\lambda_r)\right)\right]. \tag{65}$$

At the returned solution u_{r+1}, we have $R_{\gamma,r}(u_{r+1},\theta) \approx 0$. Instead of explicitly forming $D_\theta u_{r+1}$, the backward pass solves the adjoint system

$$\left[\nabla_u R_{\gamma,r}(u_{r+1},\theta)\right]^\top v_{\mathrm{adj}} = \nabla_u \mathcal{L}_{\mathrm{out}}(u_{r+1},\theta)^\top, \tag{66}$$

and then computes

$$\nabla_\theta \mathcal{L}_{\mathrm{out}} = \nabla_\theta \mathcal{L}_{\mathrm{out}}(u_{r+1},\theta) - v_{\mathrm{adj}}^\top \nabla_\theta R_{\gamma,r}(u_{r+1},\theta). \tag{67}$$

Krylov iterative methods solve the adjoint linear system using only matrix–vector products with its coefficient operator. Therefore, the backward pass only requires the two matrix-free products

$$\left[\nabla_u R_{\gamma,r}\right]^\top v_{\mathrm{adj}}, \qquad v_{\mathrm{adj}}^\top \nabla_\theta R_{\gamma,r}.$$

Let

$$y_{\gamma,r+1} = u_{r+1} - \gamma\nabla_u \psi_{\rho_r}(u_{r+1};\theta,\lambda_r), \qquad P_{U,r+1} = D\Pi_{\mathbf{U}}(y_{\gamma,r+1}).$$

When $\mathbf{U}$ is a box, $P_{U,r+1}$ is a diagonal active-set mask that retains the free components and removes the active components. Since $\mathbf{U}$ is independent of θ, differentiating Eq. (65) gives

$$\nabla_u R_{\gamma,r} = \frac{1}{\gamma}(I - P_{U,r+1}) + P_{U,r+1}\nabla_{uu}^2 \psi_{\rho_r}, \qquad \nabla_\theta R_{\gamma,r} = P_{U,r+1}\nabla_{u\theta}^2 \psi_{\rho_r}. \tag{68}$$

Therefore, the two products required by Eqs. (66) and (67) are

$$\left[\nabla_u R_{\gamma,r}\right]^\top v_{\mathrm{adj}} = \frac{1}{\gamma}(I - P_{U,r+1})^\top v_{\mathrm{adj}} + \nabla_{uu}^2 \psi_{\rho_r}\left(P_{U,r+1}^\top v_{\mathrm{adj}}\right), \tag{69}$$

and

$$v_{\mathrm{adj}}^\top \nabla_\theta R_{\gamma,r} = \left(P_{U,r+1}^\top v_{\mathrm{adj}}\right)^\top \nabla_{u\theta}^2 \psi_{\rho_r}. \tag{70}$$

Hence, it remains to evaluate the smooth-part Hessian-vector product $\nabla_{uu}^2 \psi_{\rho_r} v$ and the mixed vector-Jacobian product $q^\top \nabla_{u\theta}^2 \psi_{\rho_r}$.

Recall the reduced ALM objective in Eq. (6). At u_{r+1}, define

$$w_{r+1} = G(u_{r+1},\theta) + \rho_r^{-1}\lambda_r, \qquad z_{r+1} = \Pi_{\mathcal{D}}(w_{r+1}), \qquad \bar{\lambda}_{r+1} = \rho_r(w_{r+1} - z_{r+1}).$$

Also define

$$J_{G,r+1} = \nabla_u G(u_{r+1},\theta), \qquad P_{D,r+1} = D\Pi_{\mathcal{D}}(w_{r+1}), \qquad S_{D,r+1} = I - P_{D,r+1}.$$

For $\mathcal{D} = \{0\}^{n_c} \times \mathbb{R}_-^{n_h}$, the mask $S_{D,r+1}$ retains the equality components and the locally active inequality components, while removing the inactive inequality components.

The gradient of the reduced smooth objective is

$$\nabla_u \psi_{\rho_r}(u_{r+1};\theta,\lambda_r) = \nabla_u \ell(u_{r+1},\theta) + J_{G,r+1}^\top \bar{\lambda}_{r+1}. \tag{71}$$

Since $D_u\bar{\lambda}_{r+1}[v] = \rho_r S_{D,r+1} J_{G,r+1} v$, the Hessian-vector product satisfies, for any vector v,

$$\nabla_{uu}^2 \psi_{\rho_r} v = \nabla_{uu}^2\left(\ell + \bar{\lambda}_{r+1}^\top G\right) v + \rho_r J_{G,r+1}^\top S_{D,r+1} J_{G,r+1} v. \tag{72}$$

In the first term, $\bar{\lambda}_{r+1}$ is held fixed. Thus, this term is a Lagrangian Hessian-vector product. The second term requires only one JVP through G, one application of the active-set mask $S_{D,r+1}$, and one VJP through G.

During differentiation of the final ALM subproblem, the incoming ALM multiplier λ_r and the penalty parameter ρ_r are treated as fixed. Since $D_\theta\bar{\lambda}_{r+1}[\delta\theta] = \rho_r S_{D,r+1}\nabla_\theta G(u_{r+1},\theta)\delta\theta$, the mixed product satisfies

$$\nabla_{u\theta}^2 \psi_{\rho_r}\delta\theta = \nabla_{u\theta}^2\left(\ell + \bar{\lambda}_{r+1}^\top G\right)\delta\theta + \rho_r J_{G,r+1}^\top S_{D,r+1}\nabla_\theta G(u_{r+1},\theta)\delta\theta. \tag{73}$$

Equivalently, for any adjoint vector q,

$$q^\top \nabla^2_{u\theta}\psi_{\rho_r} = q^\top \nabla^2_{u\theta}\left(\ell + \bar{\lambda}^\top_{r+1} G\right) + \rho_r \left(S_{D,r+1} J_{G,r+1} q\right)^\top \nabla_\theta G(u_{r+1}, \theta). \tag{74}$$

Let $q = P^\top_{U,r+1} v_{\mathrm{adj}}$. Substituting Eq. (72) into Eq. (69), the adjoint operator used in the Krylov solver is

$$\left[\nabla_u R_{\gamma,r}\right]^\top v_{\mathrm{adj}} = \frac{1}{\gamma}(I - P_{U,r+1})^\top v_{\mathrm{adj}} + \nabla^2_{uu}\left(\ell + \bar{\lambda}^\top_{r+1} G\right) q + \rho_r J^\top_{G,r+1} S_{D,r+1} J_{G,r+1} q. \tag{75}$$

The smooth Hessian term and the projected-constraint term $J^\top_{G,r+1} S_{D,r+1} J_{G,r+1}$ are symmetric under the local regularity assumptions. However, their composition with the projection derivative $P_{U,r+1}$, through $q = P^\top_{U,r+1} v_{\mathrm{adj}}$, does not generally preserve the symmetry of the complete residual operator. Hence, GMRES provides the general Krylov solver. For the common case in which $\mathbf{U}$ consists of box constraints and the active set is locally identified, $P_{U,r+1}$ is a symmetric diagonal projector. Its range $\mathcal{V}_{r+1} := \mathrm{range}(P_{U,r+1})$ is the tangent space of the identified box face. Restricting the adjoint system to this subspace gives the projected operator $P_{U,r+1}\nabla^2_{uu}\psi_{\rho_r} P_{U,r+1}$. For every nonzero $d \in \mathcal{V}_{r+1}$, Eq. (33), together with the slack elimination leading to Eq. (43), gives

$$d^\top P_{U,r+1}\nabla^2_{uu}\psi_{\rho_r} P_{U,r+1} d = d^\top \nabla^2_{uu}\psi_{\rho_r} d > 0. \tag{76}$$

Thus, locally under SOSC and a sufficiently large ALM penalty, the restricted operator is symmetric positive definite, so we use CG by default. If nonpositive curvature or numerical failure is detected, the implementation automatically falls back to MINRES.

Substituting Eq. (74) into Eq. (70), the parameter-side VJP is

$$v^\top_{\mathrm{adj}} \nabla_\theta R_{\gamma,r} = q^\top \nabla^2_{u\theta}\left(\ell + \bar{\lambda}^\top_{r+1} G\right) + \rho_r \left(S_{D,r+1} J_{G,r+1} q\right)^\top \nabla_\theta G(u_{r+1}, \theta). \tag{77}$$

Eqs. (75) and (77) give the final matrix-free backward operators. Their evaluation requires only HVPs, JVPs, VJPs, and active-set masks associated with $\Pi_{\mathbf{U}}$ and $\Pi_{\mathcal{D}}$, without explicitly constructing Hessian, Jacobian, or KKT matrices.

C Experiments Details

C.1 Constrained Rosenbrock Benchmark

This section examines the sensitivity-accuracy behavior characterized by Theorem 2 and the practical strategies used to control it, thereby explaining the construction, settings, and accuracy-matching protocol of Table 2.

Experimental setup. This benchmark and the constrained OCP imitation experiments were conducted on a workstation equipped with an Intel Core i5-12600KF CPU@3.70 GHz, 32 GB of RAM, and an NVIDIA GeForce RTX 4060Ti GPU. The learnable parameter is $\theta = (\theta_0, \theta_1, \theta_2, \theta_3)$, with nominal value $(10, 1, 10^{-3}, 1.35)$, and the outer sensitivity loss is $\mathcal{L}_{\mathrm{out}}(x^\star) = \frac{1}{2}\|x^\star - x_{\mathrm{tar}}\|_2^2$, where x_{tar} is the sinusoidal target used in the benchmark. The solver settings are summarized in Table 5.

Table 5: Solver settings for the constrained Rosenbrock benchmark.

Quantity	Value
CasADi forward solver	IPOPT
CasADi sensitivity pipeline	SQP/qpOASES + sparse QR
Forward ALM/IPOPT tolerance	10^{-3}
Krylov tolerance	10^{-2}
lapanda/Explicit Krylov method	CG/MINRES
Maximum Krylov iterations	200
Maximum PANDA iterations	4000
Maximum ALM iterations	20
Initial penalty ρ_0	2
Penalty update factor	10

Following the official `CasADi` sensitivity workflow, we first solved the original NLP with IPOPT and used the converged primal solution to initialize a differentiable `sqpmethod` solver. Since this initialization already satisfied the prescribed tolerance, the SQP stage terminated after one iteration.

During this iteration, `qpOASES` computed the information required for active-set identification. `CasADi` then constructed the differentiated KKT system using the exact Lagrangian Hessian and constraint Jacobian and solved the adjoint system with its default sparse-QR sensitivity solver. We also tested `qrqp` as the QP backend of `sqpmethod`, but it occasionally failed on larger instances. For the problem sizes successfully solved by both backends, their runtime and memory overheads were similar. We therefore report results obtained with the `qpOASES`-based configuration.

Sensitivity reference and error metrics. To obtain a reference sensitivity for each solution returned by `lapanda`, we construct the KKT sensitivity system of the original problem and solve it to high accuracy. The relative linear residual is below 10^{-15} in all reported cases. Let $g = \nabla_\theta \mathcal{L}_{\text{out}}^{\text{lapanda}}$ and let g_{ref} denote the reference gradient. We report both relative error and cosine similarity:

$$\text{Err}_\nabla = \frac{\|g - g_{\text{ref}}\|_2}{\|g_{\text{ref}}\|_2}, \qquad \text{Cos}_\nabla = \frac{g^\top g_{\text{ref}}}{\|g\|_2 \|g_{\text{ref}}\|_2}. \tag{78}$$

The former measures the full gradient discrepancy, whereas the latter isolates directional agreement. Constraint violation is measured by $\max_i [g_i(x, \theta)]_+$.

Relationship between penalty and sensitivity error. The analysis in Section 4.2 gives the error order $O(\rho_r^{-1}) + O(\epsilon_{r+1})$. To examine this relation, we solve one ALM subproblem for the representative $n = 200$ instance at each fixed penalty. Table 6 shows that increasing ρ improves feasibility and gradient accuracy, while making the adjoint system more expensive.

Table 6: Penalty sweep for the $n = 200$ constrained Rosenbrock diagnostic.

Penalty ρ	Relative error ↘	Cosine similarity	Violation ↘	Backward time (ms) ↗
1	5.59×10^{-1}	0.988404	2.47×10^{-1}	0.15
3	2.91×10^{-1}	0.998718	1.28×10^{-1}	0.15
10	1.14×10^{-1}	0.999793	4.67×10^{-2}	0.18
30	4.15×10^{-2}	0.999967	1.84×10^{-2}	0.22
100	1.34×10^{-2}	0.999994	6.74×10^{-3}	0.31
300	4.80×10^{-3}	0.999999	2.32×10^{-3}	0.53
1000	6.93×10^{-4}	1.000000	7.02×10^{-4}	0.76
3000	5.64×10^{-4}	1.000000	2.34×10^{-4}	0.99

Balanced strategy. Our default strategy directly reuses the primal solution, multiplier estimate, and penalty returned by the converged forward ALM solve. The backward system therefore corresponds to the same final subproblem, with no additional tuning or forward optimization. Table 7 reports means over ten instances at each problem size. Although the relative error is about 5%–10%, the cosine similarity consistently exceeds 0.9999. Thus, the finite-penalty approximation mainly affects the gradient magnitude, while preserving its descent direction to high accuracy. This balanced strategy retains the millisecond-scale backward pass and is used in the later learning experiments, where it produces loss trajectories comparable to the baselines.

Table 7: Balanced sensitivity computation using the forward-converged penalty.

n	Final penalty	Relative error	Cosine similarity	Violation	Backward time (ms)
100	4.67×10^2	7.08×10^{-2}	0.999925	7.99×10^{-4}	0.15
200	4.59×10^2	9.42×10^{-2}	0.999910	5.75×10^{-4}	0.25
500	3.24×10^2	9.89×10^{-2}	0.999922	6.18×10^{-4}	0.62
1000	4.53×10^2	9.87×10^{-2}	0.999932	6.86×10^{-4}	1.54

Post-forward penalty refinement. When higher relative accuracy is required, we consider two post-forward refinement strategies. *Direct backward refinement* retains the converged primal-dual estimate and uses an enlarged penalty only in the backward operators. It requires no additional forward optimization, but the modified backward system is not exactly aligned with the subproblem solved in the forward pass. *Aligned subproblem refinement* instead increases the penalty, re-solves the final ALM subproblem from the converged state, and then differentiates the refined subproblem. This preserves the alignment in Theorem 2, at the cost of one additional warm-started subproblem solve. Table 8 compares the two strategies using 10ρ on the first instance at each problem size.

Table 8: Comparison of post-forward penalty-refinement strategies.

n	Forward-converged			Direct backward refinement			Aligned subproblem refinement			
	ρ	Grad. err.	Bwd. (ms)	ρ	Grad. err.	Bwd. (ms)	ρ	Grad. err.	Refine solve (ms)	Bwd. (ms)
100	4.12×10^2	7.07×10^{-2}	0.15	4.12×10^3	1.13×10^{-2}	0.26	4.12×10^3	9.29×10^{-3}	3.71	0.27
200	2.00×10^2	9.54×10^{-2}	0.23	2.00×10^3	2.42×10^{-2}	0.43	2.00×10^3	1.37×10^{-2}	4.17	0.44
500	2.27×10^2	9.97×10^{-2}	0.58	2.27×10^3	1.31×10^{-2}	1.16	2.27×10^3	1.31×10^{-2}	11.30	1.20
1000	2.36×10^2	9.96×10^{-2}	1.50	2.36×10^3	1.24×10^{-2}	3.04	2.36×10^3	1.25×10^{-2}	29.46	2.97

As shown in Table 8, aligned refinement reduces the relative error below 2% at every problem size. We therefore use it for the matched-accuracy comparison in the main text. Direct refinement also improves accuracy without an additional forward solve, but its backward system is not aligned with the forward solution and therefore lacks the same theoretical justification.

These results distinguish two use cases. The forward-converged penalty is the balanced default: its gradient direction is already highly accurate and no extra computation is required. When higher relative gradient accuracy is required, aligned refinement provides the principled option used in the main benchmark. Direct refinement remains available as a low-cost heuristic knob.

It is worth noting that the subsequent imitation-learning experiments use the forward-converged penalty by default. Although the resulting gradients have moderate relative error, their high cosine similarity with the reference gradient suggests that the update direction remains sufficiently accurate for the learning tasks considered here. Such approximate gradients can be substantially cheaper to compute, and some baselines likewise employ approximate sensitivity methods. The broadly similar loss trends observed in the subsequent experiments provide empirical support for this strategy.

Matched-accuracy comparison and explicit-KKT ablation. Table 2 uses ten instances at each problem size. Gradient accuracy is evaluated by Eq. (78) against the same high-accuracy original-NLP KKT gradient at the aligned `lapanda` point for each instance. For all three methods, the timings are averaged over five repetitions on each of the same ten instances, following an untimed warm-up at each problem size. For `lapanda`, we apply aligned refinement with 10ρ. Its reported forward time contains only the original ALM solve, whereas its backward time includes the warm-started final subproblem refinement and the subsequent matrix-free adjoint solve.

Table 9: Mean (maximum) relative gradient error (%) under the matched-accuracy protocol.

n	lapanda	Explicit KKT	CasADi
100	0.95 (1.16)	1.09 (2.32)	0.02 (0.03)
200	1.35 (1.46)	0.68 (0.74)	0.01 (0.02)
500	1.30 (1.35)	0.23 (0.23)	0.43 (0.43)
1000	1.24 (1.26)	0.33 (0.33)	< 0.01 (< 0.01)

The explicit-KKT baseline shares the unrefined `lapanda` forward result, explicitly assembles the KKT matrix of the original NLP, and solves the adjoint system with MINRES through the same C-based Krylov interface and tolerance. MINRES is used because the symmetric saddle-point KKT matrix is generally indefinite. It therefore provides a matrix-based sensitivity baseline under the same base forward solve.

Memory overhead is evaluated using the resident set size (RSS) of the solver process. We consider both the memory increase during solver construction and the temporary memory used during solution. Solver construction mainly includes problem initialization, computational-graph construction, and operator generation, whereas temporary memory accounts for intermediate variables and workspaces allocated during solving. The peak temporary increment typically occurs within the first few solves, since later calls can reuse previously allocated memory. We therefore report

$$\text{total memory} = \text{build RSS peak increment} + \text{peak solve RSS increment}. \tag{79}$$

C.2 Constrained OCP Imitation Experiments

Common setup. All OCP experiments use a prediction horizon of $N = 20$ and a sampling interval of $\Delta t = 0.05$. We adopt the single-shooting transcription described in Appendix A. Thus, the optimization variable is the stacked control sequence $u = (u_0^\top, \dots, u_{N-1}^\top)^\top$, whereas the state trajectory is obtained by recursively rolling out the dynamics. The teacher trajectory is generated using the target parameter θ_{true}, and the learner minimizes the imitation loss in Eq. (16).

CartPole. For the CartPole task, the system state is defined as $x_k = (p_k, \dot{p}_k, \phi_k, \dot{\phi}_k)$, where p_k and $\dot{p}_k$ denote the cart position and velocity, while ϕ_k and $\dot{\phi}_k$ denote the pole angle and angular velocity, respectively. The control input, denoted by F_k, is the horizontal force. Using forward Euler discretization, the CartPole dynamics are given by

$$\begin{aligned}
\ddot{\phi}_k &= \frac{g\sin\phi_k - \frac{\cos\phi_k\left(F_k + m_p l\dot{\phi}_k^2\sin\phi_k\right)}{m_c+m_p}}{l\left(\frac{4}{3} - \frac{m_p\cos^2\phi_k}{m_c+m_p}\right)}, \\
\ddot{p}_k &= \frac{F_k + m_p l\left(\dot{\phi}_k^2\sin\phi_k - \ddot{\phi}_k\cos\phi_k\right)}{m_c+m_p}, \\
p_{k+1} &= p_k + \Delta t\,\dot{p}_k, \\
\dot{p}_{k+1} &= \dot{p}_k + \Delta t\,\ddot{p}_k, \\
\phi_{k+1} &= \phi_k + \Delta t\,\dot{\phi}_k, \\
\dot{\phi}_{k+1} &= \dot{\phi}_k + \Delta t\,\ddot{\phi}_k.
\end{aligned} \tag{80}$$

where m_c and m_p denote the masses of the cart and pole, respectively, l is the pole half-length, g is the gravitational acceleration, and Δt is the discretization step.

We consider a finite-horizon planning problem that steers the CartPole system from an initial state ξ toward the upright equilibrium $x_{\text{tar}} = (p_{\text{tar}}, \dot{p}_{\text{tar}}, \phi_{\text{tar}}, \dot{\phi}_{\text{tar}}) = (0, 0, 0, 0)$. Define the learnable parameter vector $\theta = (q_p, q_\phi, q_{\dot{p}}, q_{\dot{\phi}}, r_F)$ and $Q_{\text{cp}}(\theta) = \text{diag}(\theta_1, \theta_3, \theta_2, \theta_4) = \text{diag}(q_p, q_{\dot{p}}, q_\phi, q_{\dot{\phi}})$ and $\|z\|^2_{Q_{\text{cp}}} = z^\top Q_{\text{cp}} z$. The OCP is

$$\min_{F_{0:N-1}} \quad \sum_{k=0}^{N-1}\left(\|x_k - x_{\text{tar}}\|^2_{Q_{\text{cp}}} + r_F F_k^2\right) + 10\|x_N - x_{\text{tar}}\|^2_{Q_{\text{cp}}} \tag{81a}$$

$$\text{s.t.} \quad x_0 = \xi, \tag{81b}$$

$$x_{k+1} = f_{\text{cp}}(x_k, F_k), \qquad k = 0, \ldots, N-1, \tag{81c}$$

$$-2 \le F_k \le 6, \qquad k = 0, \ldots, N-1, \tag{81d}$$

$$\sum_{k=0}^{N-1} F_k^2 \le 70. \tag{81e}$$

Here, f_{cp} denotes the discrete dynamics defined in Eq. (80), with $m_c = 1.0$, $m_p = 0.1$, $l = 0.5$, $g = 9.81$, and $\Delta t = 0.05$. Here, q_p, q_ϕ, $q_{\dot{p}}$, and $q_{\dot{\phi}}$ respectively weight the cart position, pole angle, cart velocity, and pole angular-velocity tracking errors, and r_F penalizes the control effort.

For open-loop imitation, multiple initial states are sampled to evaluate the robustness of the learned solution. The initial state $\xi = (p_0, \dot{p}_0, \phi_0, \dot{\phi}_0)$ is sampled according to

$$\begin{aligned}
p_0 &\sim \text{Unif}[-0.5, 0.5], \quad \dot{p}_0 \sim \text{Unif}[-0.5, 0.5], \\
\phi_0 &\sim \text{Unif}[-\pi, \pi], \qquad \dot{\phi}_0 \sim \text{Unif}[-1, 1].
\end{aligned} \tag{82}$$

Planar quadrotor. For the planar quadrotor task, the system state is defined as $x_k = (p_{x,k}, p_{z,k}, v_{x,k}, v_{z,k}, \alpha_k, \omega_k)$, where $p_{x,k}$ and $p_{z,k}$ denote the horizontal and vertical positions, $v_{x,k}$ and $v_{z,k}$ denote the corresponding velocities, and α_k and ω_k denote the attitude angle and angular velocity, respectively. The control input is $u_k = (T_k, \tau_k)$, where T_k is the collective thrust and τ_k is the rotational control input. The discrete dynamics are given by

$$\begin{aligned}
v_{x,k+1} &= v_{x,k} - \Delta t\, T_k \sin\alpha_k, \\
v_{z,k+1} &= v_{z,k} + \Delta t\,(T_k\cos\alpha_k - g), \\
\omega_{k+1} &= \omega_k + 4\Delta t\,\tau_k, \\
\alpha_{k+1} &= \alpha_k + \Delta t\,\omega_{k+1}, \\
p_{x,k+1} &= p_{x,k} + \Delta t\, v_{x,k+1}, \\
p_{z,k+1} &= p_{z,k} + \Delta t\, v_{z,k+1}.
\end{aligned} \tag{83}$$

We consider a finite-horizon planning problem that steers the quadrotor from the initial state ξ toward the target state $x_{\text{tar}} = (1.0, 1.2, 0, 0, 0, 0)$. Define $Q_{\text{quad}} = \text{diag}(q_p, q_p, q_v, q_v, q_\alpha, q_\omega)$, $R_{\text{quad}} = \text{diag}(r_T, r_\tau)$, and $u_{\text{hov}} = (g, 0)$. The OCP is

$$\begin{aligned}
\min_{T_{0:N-1},\, \tau_{0:N-1}} \quad & \sum_{k=0}^{N-1} \Big(\|x_k - x_{\text{tar}}\|^2_{Q_{\text{quad}}} + \|u_k - u_{\text{hov}}\|^2_{R_{\text{quad}}} \Big) && \text{(84a)} \\
& \quad + 15\|x_N - x_{\text{tar}}\|^2_{Q_{\text{quad}}} \\
\text{s.t.} \quad & x_0 = \xi, && \text{(84b)} \\
& x_{k+1} = f_{\text{quad}}(x_k, T_k, \tau_k), \qquad k = 0, \dots, N-1, && \text{(84c)} \\
& 0 \le T_k \le 2.2g, \qquad -3 \le \tau_k \le 3, \qquad k = 0, \dots, N-1, && \text{(84d)} \\
& 0.43 \le p_{z,k} \le 1.20, \qquad p_{x,k} \le 1.0, \qquad k = 0, \dots, N, && \text{(84e)} \\
& |\alpha_k| \le 0.34, \qquad k = 0, \dots, N. && \text{(84f)}
\end{aligned}$$

Here, f_{quad} denotes the discrete dynamics defined in Eq. (83), with $g = 9.81$ and $\Delta t = 0.05$. The thrust regularization term is centered at $T_k = g$, corresponding to the nominal hovering thrust of the normalized model. The learnable parameter vector is $\theta = (q_p, q_v, q_\alpha, q_\omega, r_T, r_\tau)$, where q_p, q_v, q_α, and q_ω respectively weight the position, velocity, attitude-angle, and angular-velocity tracking errors, while r_T and r_τ penalize the thrust and rotational control inputs.

For open-loop imitation, the initial state is sampled around $\bar{x}_0 = (-1.0, 0.65, 0, 0, 0.15, 0)$:

$$\begin{aligned}
p_{x,0} &= \bar{p}_{x,0} + \text{Unif}[-0.25, 0.25], \quad p_{z,0} = \bar{p}_{z,0} + \text{Unif}[-0.10, 0.10], \\
v_{x,0} &= \bar{v}_{x,0} + \text{Unif}[-0.10, 0.10], \quad v_{z,0} = \bar{v}_{z,0} + \text{Unif}[-0.10, 0.10], \\
\alpha_0 &= \bar{\alpha}_0 + \text{Unif}[-0.08, 0.08], \qquad \omega_0 = \bar{\omega}_0 + \text{Unif}[-0.10, 0.10].
\end{aligned} \tag{85}$$

Two-link robot arm. For the two-link robot-arm task, the system state is defined as $q_k = (q_{1,k}, q_{2,k})$, where $q_{1,k}$ and $q_{2,k}$ denote the two joint angles. The control input is $u_k = (\dot{q}_{1,k}, \dot{q}_{2,k})$, where $\dot{q}_{1,k}$ and $\dot{q}_{2,k}$ are the commanded joint velocities. The discrete joint dynamics and the corresponding end-effector position are given by

$$\begin{aligned}
q_{k+1} &= q_k + \Delta t\, u_k, \\
p_{\text{ee}}(q_k) &= \begin{bmatrix} \cos q_{1,k} + 0.8\cos(q_{1,k} + q_{2,k}) \\ \sin q_{1,k} + 0.8\sin(q_{1,k} + q_{2,k}) \end{bmatrix}.
\end{aligned} \tag{86}$$

We consider a finite-horizon planning problem that steers the robot arm from an initial joint configuration ξ toward the target configuration $q_{\text{tar}} = (0.75, -0.65)$ while avoiding a circular obstacle in the end-effector workspace. Define

$$\ell_{\text{arm}}(q) = q_q\|q - q_{\text{tar}}\|_2^2 + q_{\text{ee}}\|p_{\text{ee}}(q) - p_{\text{ee}}(q_{\text{tar}})\|_2^2, \qquad R_{\text{arm}} = \text{diag}(r_1, r_2).$$

The OCP is

$$\begin{aligned}
\min_{u_{0:N-1}} \quad & \sum_{k=0}^{N-1} \big(\ell_{\text{arm}}(q_k) + \|u_k\|^2_{R_{\text{arm}}} \big) + 20\ell_{\text{arm}}(q_N) && \text{(87a)} \\
\text{s.t.} \quad & q_0 = \xi, && \text{(87b)} \\
& q_{k+1} = q_k + \Delta t\, u_k, \qquad k = 0, \dots, N-1, && \text{(87c)} \\
& -3 \le \dot{q}_{1,k} \le 3, \qquad -3 \le \dot{q}_{2,k} \le 3, \qquad k = 0, \dots, N-1, && \text{(87d)} \\
& -1.10 \le q_{1,k} \le 0.75, \qquad -0.65 \le q_{2,k} \le 1.28, \qquad k = 1, \dots, N, && \text{(87e)} \\
& (0.29 + m_{\text{obs}})^2 - \left\| p_{\text{ee}}(q_k) - \begin{bmatrix} 1.30 \\ 0.30 \end{bmatrix} \right\|_2^2 \le 0, \qquad k = 1, \dots, N. && \text{(87f)}
\end{aligned}$$

Here, the dynamics are defined in Eq. (86), with $\Delta t = 0.05$. The obstacle is centered at $p_{\text{obs}} = (1.30, 0.30)$, and the constraint in Eq. (87f) requires the end effector to remain outside a circle with radius $0.29 + m_{\text{obs}}$. The learnable parameter vector is $\theta = (q_q, q_{\text{ee}}, r_1, r_2, m_{\text{obs}})$, where q_q weights the joint-configuration tracking error, q_{ee} weights the end-effector tracking error, r_1 and r_2 penalize the two commanded joint velocities, and m_{obs} represents the learnable obstacle-safety margin.

For open-loop imitation, the initial joint configuration is sampled according to

$$q_{1,0} \sim \text{Unif}[-1.10, -0.80], \qquad q_{2,0} \sim \text{Unif}[1.05, 1.28]. \tag{88}$$

Open-loop and closed-loop imitation protocols.

Open-loop imitation. For each task, we sample $N_{\text{demo}} = 32$ initial conditions $\{\xi_j\}_{j=1}^{N_{\text{demo}}}$ according to the task-specific distributions in Eqs. (82), (85), and (88). Starting from each ξ_j, the OCP is first solved using the teacher parameter θ_{true}, producing the demonstration trajectories $(X_j^{\text{demo}}, U_j^{\text{demo}})$. During learning, the same OCP is repeatedly solved from the same initial condition using the current parameter θ, yielding $(X^\star(\theta;\xi_j), U^\star(\theta;\xi_j))$. The parameter is then updated by minimizing the imitation objective in Eq. (16) over all 32 initial conditions.

Closed-loop imitation. For closed-loop imitation, a reference rollout is first generated using θ_{true}. At each MPC instant, the OCP is solved from the current teacher state, only the first element of the optimized control trajectory is applied, and the system is propagated to the next state. This procedure is repeated for 50 MPC steps, producing a closed-loop reference trajectory and the corresponding sequence of optimal predictions $\{(X_t^{\text{demo}}, U_t^{\text{demo}})\}_{t=0}^{49}$. The learner performs the same receding-horizon rollout using the current parameter θ, producing $\{(X_t^\star(\theta), U_t^\star(\theta))\}_{t=0}^{49}$. At every MPC instant, the discrepancy between the learner and teacher state-control predictions is evaluated using the same form as Eq. (16), and the losses are accumulated over the complete rollout.

Hyperparameter settings. Table 10 summarizes the learning and `lapanda` solver configurations. Here, ε_{in} and ε_{ALM} are the PANDA stationarity and ALM feasibility tolerances, while ρ_0 and τ_ρ are the initial penalty and its update factor. The matrix-free backward pass uses the forward-converged penalty directly and solves the reduced system by CG, with an automatic MINRES fallback if CG detects nonpositive curvature.

Table 10: Learning configurations and lapanda hyperparameters for the OCP imitation experiments.

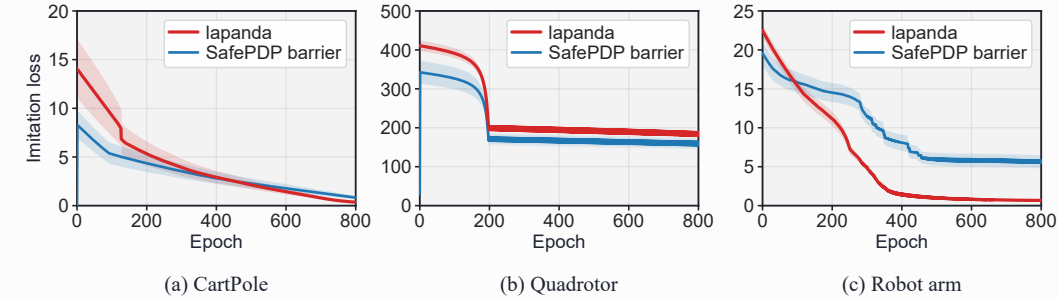
Protocol	OCP	Epochs	Learning rate	$\varepsilon_{\text{in}}/\varepsilon_{\text{ALM}}$	(ρ_0, τ_ρ)	Max iter. (PANDA/ALM)	Warm start
Open-loop	CartPole	800	2×10^{-3}	$10^{-3}/10^{-3}$	$(10, 5)$	1500/20	Previous epoch
	Quadrotor	800	2×10^{-3}	$10^{-3}/10^{-3}$	$(10, 5)$	1500/20	Previous epoch
	Robot Arm	800	2×10^{-3}	$10^{-3}/10^{-3}$	$(10^3, 5)$	1500/20	Previous epoch
Closed-loop	CartPole	500	1×10^{-3}	$10^{-3}/10^{-3}$	$(10, 5)$	1500/20	Previous MPC step
	Quadrotor	500	1×10^{-3}	$10^{-3}/10^{-3}$	$(10, 5)$	1500/20	Previous MPC step
	Robot Arm	500	1×10^{-3}	$10^{-3}/10^{-3}$	$(10^3, 5)$	1500/20	Previous MPC step

The other differentiable OCP solvers use the same prediction horizon, sampling interval, learning schedule, and stopping tolerance as the corresponding protocol in Table 10.

Table 11: Method-specific solver configurations of the differentiable OCP baselines.

Method	Forward pass	Backward pass	Maximum iterations
SafePDP	IPOPT-based constrained OCP solve	Constrained auxiliary system (COC)	IPOPT: 3000
TurboMPC-CPU	SQP-ADMM with `admm_jax_loop_pcg`	`admm_jax_loop_pcg`	SQP: 50; ADMM: 1000
TurboMPC-GPU	SQP-ADMM with `admm_fused_cudss`	`direct_cudss_ffi`	SQP: 50; ADMM: 1000

For `SafePDP`, we evaluate its constrained auxiliary-system sensitivity mode (`coc`) and barrier approximation with $\gamma_{\text{bar}} = 10^{-2}$. The barrier variant exhibits an abrupt initial change and a less stable imitation-loss trajectory, as shown in Fig. 7; hence, the main paper reports the `coc` results.



(a) CartPole (b) Quadrotor (c) Robot arm

Figure 7: Imitation loss comparison between SafePDP-barrier and lapanda.

For `TurboMPC`, we report the warm-started configuration for each hardware backend as `TurboMPC-CPU` and `TurboMPC-GPU`. Both use the same SQP and ADMM iteration limits and the full Hessian, differing only in their CPU- and GPU-specific linear-algebra backends.

Sensitivity accuracy over learning. Table 12 evaluates the backward approximation at representative learning checkpoints. At each sampled problem, we keep the `lapanda` forward solution fixed and use a high-accuracy solve of the original KKT sensitivity system at the same point as the reference. The reported metrics are computed from the gradients averaged over the 32 open-loop scenarios or the 50 MPC instants of the closed-loop rollout.

Table 12: Sensitivity accuracy of `lapanda` over the OCP imitation-learning experiments.

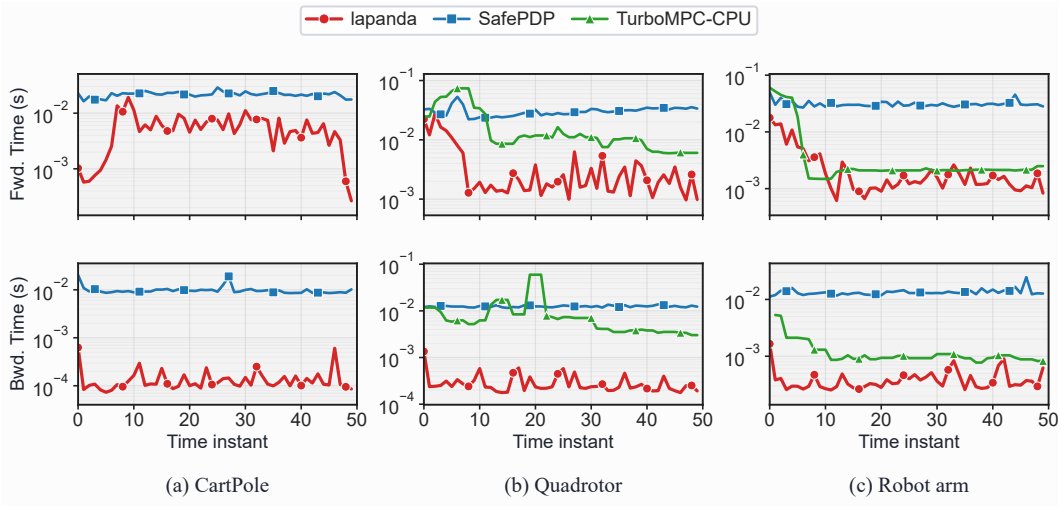
OCP	Metric	Open-loop epoch				Closed-loop epoch			
		0	200	400	800	0	125	250	500
CartPole	Rel. err. (%)	0.012	0.004	0.012	0.010	0.002	0.006	0.006	0.007
	Cos. sim.	0.99999999	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
Quadrotor	Rel. err. (%)	0.048	0.286	0.363	1.306	0.058	0.137	0.134	0.076
	Cos. sim.	0.99999997	0.99999999	0.99999998	0.99999982	0.99999992	0.99999999	1.00000000	1.00000000
Robot Arm	Rel. err. (%)	8.459	0.076	0.688	0.027	1.178	4.572	0.084	0.136
	Cos. sim.	0.99643301	0.99999972	0.99999237	1.00000000	0.99993896	0.99901230	0.99999970	0.99999995

Across the learning checkpoints, the relative sensitivity error is generally small, with larger deviations confined to a few Robot Arm checkpoints with less favorable numerical conditioning. More importantly, the cosine similarity remains consistently above 0.996, indicating that the approximate gradients preserve the reference gradient directions even when their magnitudes are less accurate. This strong directional agreement helps explain why, in Figs. 3 and 4, `lapanda` produces loss trajectories that are nearly identical to those of the corresponding baselines, even when using the balanced sensitivity strategy.

Timing and memory measurement. For the runtime comparison, all forward and backward computation times are arithmetic means of wall-clock time over all recorded solves. For `lapanda` and `SafePDP`, the forward and backward procedures can be timed separately. In contrast, `TurboMPC` does not expose an independent backward call. We therefore first measure a forward-only solve and then measure the complete value-and-gradient evaluation. Its backward time is estimated as

$$t_{\text{bwd}}^{\text{TurboMPC}} = t_{\text{value+grad}}^{\text{TurboMPC}} - t_{\text{fwd}}^{\text{TurboMPC}}. \tag{89}$$

In addition to the averaged runtime results, we record the forward and backward times at each MPC instant during one representative closed-loop rollout, as shown in Fig. 8. The solvers exhibit different runtime profiles over the rollout. In particular, the computation time of `lapanda` generally decreases after the initial MPC steps. This trend is consistent with the benefit of warm-starting: the primal variables, multipliers, penalty parameters, and shifted control trajectory obtained at the previous MPC instant provide an increasingly accurate initialization for the subsequent problem.



(a) CartPole (b) Quadrotor (c) Robot arm

Figure 8: Computation times at each MPC instant during a representative closed-loop rollout.

Scaling with the horizon length. To evaluate solver performance across different horizon lengths N, we vary N up to 200 for the Quadrotor OCP while fixing the total prediction time at 2.5 s. Thus, different values of N correspond to different temporal discretization resolutions of the same prediction window. For each N, we report the mean total forward-and-backward time over a 30-step warm-started MPC rollout. We use `TurboMPC-GPU` as a representative GPU-based baseline.

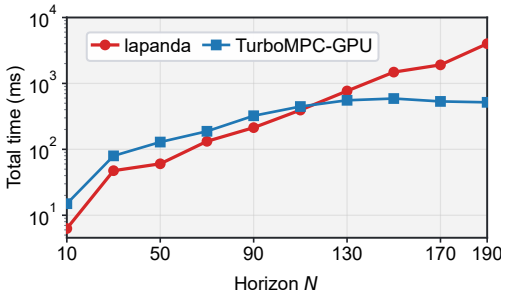


Figure 9: Mean solve time versus horizon length.

As shown in Fig. 9, `lapanda` is faster for smaller N, whereas `TurboMPC-GPU` becomes more advantageous as N increases. This result clarifies the intended application regime of our method: `lapanda` is particularly suitable for solving complex short-horizon problems on resource-constrained platforms, while specialized GPU solvers are preferable for large-horizon problems when dedicated GPU resources are available.

C.3 Embedded Obstacle-Avoidance Experiments

Embedded platform. The embedded experiments are conducted on an NVIDIA Jetson Orin Nano platform equipped with a six-core Arm Cortex-A78AE v8.2 CPU, a 1024-core NVIDIA Ampere GPU with 32 Tensor Cores, 8 GB of 128-bit LPDDR5 memory. The reported `lapanda` experiments execute the exported standalone C solver on the Arm CPU.

Circular-obstacle task. We first consider a smooth nonlinear obstacle-avoidance problem. The vehicle is described by a kinematic bicycle model with state $x_k = (p_{x,k}, p_{y,k}, \psi_k)$ and control input $u_k = (v_k, \delta_k)$, where $(p_{x,k}, p_{y,k})$ is the planar position, ψ_k is the heading angle, v_k is the longitudinal velocity, and δ_k is the steering angle. The discrete dynamics are

$$\begin{aligned} p_{x,k+1} &= p_{x,k} + \Delta t\, v_k \cos\psi_k, \\ p_{y,k+1} &= p_{y,k} + \Delta t\, v_k \sin\psi_k, \\ \psi_{k+1} &= \psi_k + \frac{\Delta t}{L} v_k \tan\delta_k, \end{aligned} \tag{90}$$

where horizon $N = 12$, $\Delta t = 0.12$, and $L = 0.45$.

Let $p_k = (p_{x,k}, p_{y,k})$ denote the vehicle position. The full differentiable problem-parameter vector is $\theta_{\text{circ}} = (q_p, q_\psi, r_v, r_\delta, q_f, r_o, c_y)$, where $c_{\text{circ}} = (0, c_y) \in \mathbb{R}^2$ and $r_{\text{circ}}(\theta_{\text{circ}}) = r_o$ denote the center and radius of the circular obstacle, respectively. The smooth obstacle constraint is

$$h_k^{\text{circ}}(x_k; \theta_{\text{circ}}) := r_{\text{circ}}(\theta_{\text{circ}})^2 - \|p_k - c_{\text{circ}}\|_2^2 \le 0. \tag{91}$$

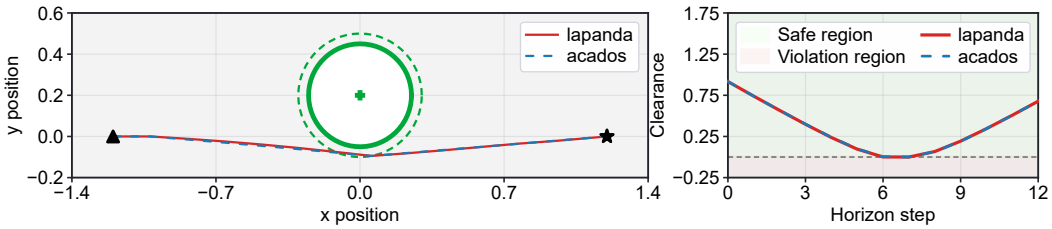
Starting from $x_{\text{init}} = (-1.2, 0, 0)$, the vehicle is required to reach $x_{\text{tar}} = (1.2, 0, 0)$ while remaining outside the circular obstacle. The corresponding OCP is

$$\begin{aligned} \min_{u_{0:N-1}} \quad & \sum_{k=0}^{N-1} \left[q_p \|p_{k+1} - p_{\text{tar}}\|_2^2 + q_\psi (\psi_{k+1} - \psi_{\text{tar}})^2 + r_v v_k^2 + r_\delta \delta_k^2 \right] && \text{(92a)} \\ & + q_f \left[q_p \|p_N - p_{\text{tar}}\|_2^2 + q_\psi (\psi_N - \psi_{\text{tar}})^2 \right] \\ \text{s.t.} \quad & x_0 = x_{\text{init}}, && \text{(92b)} \\ & x_{k+1} = f_{\text{veh}}(x_k, u_k), \qquad k = 0, \dots, N-1, && \text{(92c)} \\ & -1.5 \le v_k \le 1.5, \qquad -25^\circ \le \delta_k \le 25^\circ, \qquad k = 0, \dots, N-1, && \text{(92d)} \\ & h_k^{\text{circ}}(x_k; \theta_{\text{circ}}) \le 0, \qquad k = 1, \dots, N. && \text{(92e)} \end{aligned}$$

Here, we use $\theta_{\text{circ}} = (10.0, 0.2, 10^{-2}, 10^{-2}, 30.0, 0.30, 0.20)$ and f_{veh} denotes the dynamics in Eq. (90). Given a teacher control trajectory U^{teacher}, the outer learning objective is

$$\mathcal{L}_{\text{circ}}(\theta_{\text{circ}}) = \frac{1}{2} \left\| U^\star(\theta_{\text{circ}}) - U^{\text{teacher}} \right\|_2^2. \tag{93}$$

For the embedded timing benchmark, we set $U^{\text{teacher}} = 0$; a nonzero teacher changes the adjoint right-hand side but not the solver configuration. Both `lapanda` and `acados` successfully solve the circular-obstacle task. Figure 10 shows the resulting vehicle trajectories and the corresponding obstacle clearances.



(a) Trajectory (b) Obstacle clearance

Figure 10: Planned trajectory for the circular-obstacle avoidance task on the embedded platform.

Rectangular-obstacle task. The rectangular task uses the same vehicle model, initial and target states, and planning objective as in Eq. (92), but adopts the horizon $N = 20$, velocity bounds $-1 \le v_k \le 1$, and steering-angle bounds $-0.7 \le \delta_k \le 0.7$ rad. For this task, the PANDA and ALM tolerances are set to 10^{-3} and 10^{-4}, respectively. Its obstacle constraint is defined by

$$x_{\min} = -0.35 - m_l, \qquad x_{\max} = 0.35 + m_r, \qquad y_{\min} = -0.22 - m_b, \qquad y_{\max} = 0.22 + m_t.$$

A point lies outside the rectangle if at least one of the following conditions holds:

$$p_{x,k} \le x_{\min} \quad \vee \quad p_{x,k} \ge x_{\max} \quad \vee \quad p_{y,k} \le y_{\min} \quad \vee \quad p_{y,k} \ge y_{\max}. \tag{94}$$

Thus, the feasible region is a nonconvex union of four half-spaces rather than a single smooth inequality set. To encode this logical disjunction, define the nonnegative penetration terms

$$s_{\mathrm{L},k} = \max(p_{x,k} - x_{\min}, 0), \quad s_{\mathrm{R},k} = \max(x_{\max} - p_{x,k}, 0),$$
$$s_{\mathrm{B},k} = \max(p_{y,k} - y_{\min}, 0), \quad s_{\mathrm{T},k} = \max(y_{\max} - p_{y,k}, 0).$$

When the vehicle is outside the rectangle, at least one of these terms is zero. When it is strictly inside the rectangle, all four terms are positive. Consequently, the disjunction in Eq. (94) can be represented by the complementarity-style constraint

$$\chi_{\text{rect}}(x_k; \vartheta_{\text{rect}}) := \frac{1}{2} s_{\mathrm{L},k}^2 s_{\mathrm{R},k}^2 s_{\mathrm{B},k}^2 s_{\mathrm{T},k}^2 = 0. \tag{95}$$

This product constraint is an MPCC-style representation of the logical "or" relation: it is zero whenever at least one separating condition is satisfied and positive only when the vehicle lies inside the obstacle. We therefore impose Eq. (95) directly as a nonlinear equality constraint.

The full problem-parameter vector supplied to the solver and its learnable subset are

$$\vartheta_{\text{rect}} = (q_p, q_\psi, r_v, r_\delta, q_f, m_l, m_r, m_b, m_t), \qquad \theta_{\text{rect}} = (m_l, m_r, m_b).$$

We fix $(q_p, q_\psi, r_v, r_\delta, q_f) = (5.0, 0.2, 10^{-2}, 10^{-2}, 20.0)$ and $m_t = 0.220$. Thus,

$$\vartheta_{\text{rect}}(\theta_{\text{rect}}) = (5.0, 0.2, 10^{-2}, 10^{-2}, 20.0, m_l, m_r, m_b, 0.220).$$

The teacher and initial full obstacle-margin vectors are

$$m^{\text{teacher}} = (0.120, 0.120, 0.010, 0.220), \qquad m^{\text{init}} = (0.150, 0.150, 0.160, 0.220).$$

The rectangular-obstacle imitation loss is

$$\mathcal{L}_{\text{rect}}(\theta_{\text{rect}}) = \frac{1}{2} \left\| U^\star(\vartheta_{\text{rect}}(\theta_{\text{rect}})) - U^{\text{teacher}} \right\|_2^2. \tag{96}$$

Embedded solver configuration. The principal solver settings used in the embedded experiments are summarized in Table 13. The task definitions, horizon, control bounds, obstacle geometry, and learnable parameters have been specified above and are therefore omitted from the table.

Table 13: Solver settings for the embedded obstacle-avoidance experiments.

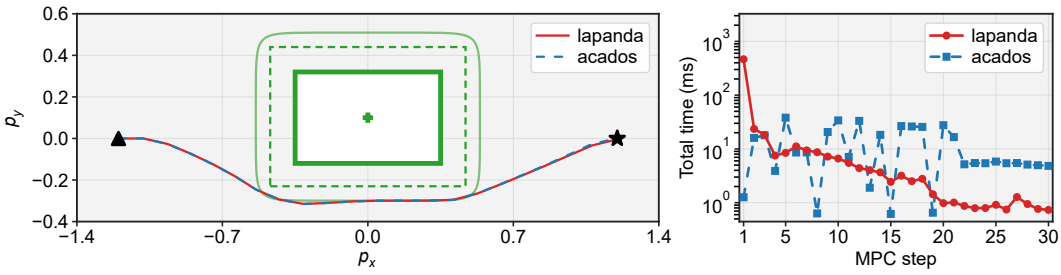
Task	Method	Hessian treatment	Maximum iterations	Forward tol.	ALM settings
Circle	lapanda	–	PANDA/ALM: 2000/100	$10^{-1}/(2\times10^{-3})$	$\rho_0 = 10^4, \tau_\rho = 10$
	acados	EXACT + MIRROR	NLP/QP: 1000/200	2×10^{-3}	–
Rectangle	lapanda	–	PANDA/ALM: 2000/100	$10^{-3}/10^{-4}$	$\rho_0 = 10^4, \tau_\rho = 10$
	acados	Gauss-Newton + MIRROR	NLP/QP: 1000/50	10^{-4}	–

For the rectangular task, we tested Gauss-Newton and exact-Hessian models in `acados`, enabled `MIRROR` regularization, increased the NLP iteration limit to 1000, and used a separate sensitivity-enabled solver. Nevertheless, `acados` repeatedly returned a failure status and the imitation loss remained nearly unchanged. This behavior is consistent with the local degeneracy of Eq. (95). This experiment should thus be interpreted as an empirical stress test outside the regularity assumptions of our local theory.

Smoothed rectangular-obstacle benchmark. To complement the MPCC experiment and provide a more comprehensive evaluation of `lapanda`, we additionally consider a smoothed version of the rectangular obstacle-avoidance problem. Specifically, we replace the disjunction by the conservative smooth maximum

$$\widetilde{\chi}_{\tau_{\mathrm{sm}}}(p_k) = \tau_{\mathrm{sm}} \log\left(\frac{1}{4}\sum_{i=1}^{4} \exp\left(\frac{d_i(p_k)}{\tau_{\mathrm{sm}}}\right)\right) \geq 0, \tag{97}$$

where $d(p_k) = (x_{\min} - p_{x,k},\, p_{x,k} - x_{\max},\, y_{\min} - p_{y,k},\, p_{y,k} - y_{\max})$. We use $\tau_{\mathrm{sm}} = 0.05$, which rounds the four corners while conservatively preserving the obstacle. Since `acados` still fails to complete the task from a zero-control initialization, we manually construct a feasible nonzero trajectory for its initial solve and shift the resulting solution in subsequent MPC steps. As shown in Fig. 11, both `acados` and `lapanda` successfully avoid the smoothed obstacle, and their computation times become comparable during the closed-loop rollout. In particular, the computation time of `lapanda` progressively decreases as the shifted warm start becomes effective and eventually stabilizes at a similar millisecond-scale level. These results demonstrate the competitive computational performance of `lapanda` on the smoothed obstacle-avoidance problem.



(a) Closed-loop trajectory (b) Per-step computation time

Figure 11: Solution trajectories and computation times for the smoothed rectangular-obstacle task.

D Implementation Details

This section briefly describes the software engineering design of `lapanda`, including its multi-platform implementation, computational optimizations, and user-oriented interfaces.

Multi-platform support. As illustrated in Fig. 12, the core algorithms of `lapanda` are implemented in C to efficiently execute the computationally intensive optimization and sensitivity routines. Python and MATLAB interfaces are provided through `Pybind11` and `MEX`, respectively. On both platforms, users define the parametric optimization problem using `CasADi`, while the interfaces generate the problem-dependent operators and invoke the C backend for forward and backward computation.

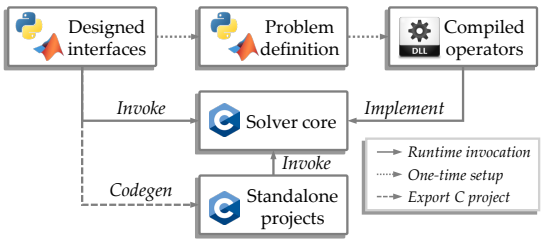


Fig. 12: Software architecture and multi-platform workflow of lapanda.

In addition, we provide an export utility on the Python platform that automatically packages a specified problem into a minimal standalone CMake project. This enables the same optimization problem to be deployed on embedded C platforms.

Engineering optimizations. The implementation incorporates two main engineering optimizations. First, once a solve is initiated, the ALM outer iterations, `PANDA` inner iterations, matrix-free operator evaluations, and backward sensitivity iterations are carried out in the C backend. The high-level interfaces mainly handle problem definition, solver configuration, data transfer, and function invocation, thereby reducing repeated language-boundary communication.

Second, we reuse the problem-dependent operators across repeated solver calls. A considerable one-time cost arises from constructing the `CasADi` computational graph and generating the corresponding objective, constraint, and derivative operators. Once the problem structure is fixed, subsequent solves require only evaluations of these pre-generated operators with updated states, parameters, and warm-start variables. We therefore compile the generated operators into a shared library that can be loaded and reused by the C solver, avoiding repeated graph construction and code generation during learning or MPC rollouts and reducing solver initialization time and memory overhead.

User-oriented interfaces. The Python and MATLAB interfaces hide operator generation, compilation, and C-backend configuration: users define the parametric problem and numerical inputs, and the solver returns $u^\star$ and its gradient with respect to θ, while `variable` denotes inputs excluded from differentiation. Representative workflows are shown below.

```python
# 1. Define a parametric optimization problem
u = ca.SX.sym("u", n_u)
theta = ca.SX.sym("theta", n_theta)
variable = ca.SX.sym("variable", n_v)

problem = CasadiProblem(
  u=u, theta=theta, variable=variable,
  cost=..., constraints=..., outer_loss=...)

# 2. Generate operators and build the solver
solver = build_solver(problem, name="demo_problem")

# 3. Solve and differentiate
result = solver.solve_alm(
  x0=x0, theta=theta_value, variable=variable_value,
  constraint_lower=..., constraint_upper=...)

u_star = result["solution"]
grad_theta = result["grad_theta"]
```

Listing 1: Python usage.

The MATLAB interface follows a similar workflow: the user defines the `CasADi` problem, creates the generated MEX solver, and invokes the forward and backward computations.

```
% 1. Define a parametric optimization problem
u = SX.sym('u', n_u, 1);
theta = SX.sym('theta', n_theta, 1);
variable = SX.sym('variable', n_v, 1);

problem.u = u;
problem.theta = theta;
problem.variable = variable;
problem.cost = ...;
problem.constraints = ...;
problem.outer_loss = ...;

% 2. Generate operators and build the solver
solver = lapanda_create_solver( ...
  problem, output_directory, 'demo_problem');

% 3. Solve and differentiate
result = solver.solve_alm( ...
  x0, theta_value, variable_value, ...
  constraint_lower, constraint_upper);

u_star = result.solution;
grad_theta = result.grad_theta;
```

Listing 2: MATLAB usage.

For embedded deployment, the Python interface can export a standalone CMake project containing the required problem-dependent operators, solver source files, and build scripts. Users can then add application-specific logic to the exported project, compile it, and run the resulting executable.

```
from lapanda import export_c_project
export_c_project(
  problem,
  out_dir="exports/demo_problem",
  name="demo_problem",
)
```

Listing 3: Exporting a problem for standalone C deployment.

```
#include "static_casadi_oracle.h"
#include "lapanda_generated_config.h"
int main(void) {
  alm_problem problem;
  double theta[LAPANDA_NTHETA] = {...}, variable[LAPANDA_NVAR] = {...};
  double u[LAPANDA_N] = {...}, grad_theta[LAPANDA_NTHETA];
  ...

  // Initialize the generated operators with backward enabled.
  backward_params.enable = 1;
  lapanda_static_init_alm_problem(&problem, constraint_lower, constraint_upper,
    &solver_params, &backward_params);

  // The solution and gradient are returned in u and grad_theta.
  alm_solve_with_backward(&problem, &alm_params, u,
    multipliers, theta, variable, &info,
    grad_theta, &backward_info);
  return 0;
}
```

Listing 4: Standalone C usage.