# RoboKit: Accelerating Parallel Differentiable Optimization for Robot Learning

Jianglong Ye* †, Xuanbin Peng*, Conor Mc Gartoll*, Changwei Jing*
Lai Wei, Jiali Chen, Xueyan Zou, Xiaolong Wang
UC San Diego

https://openrobokit.com/robokit

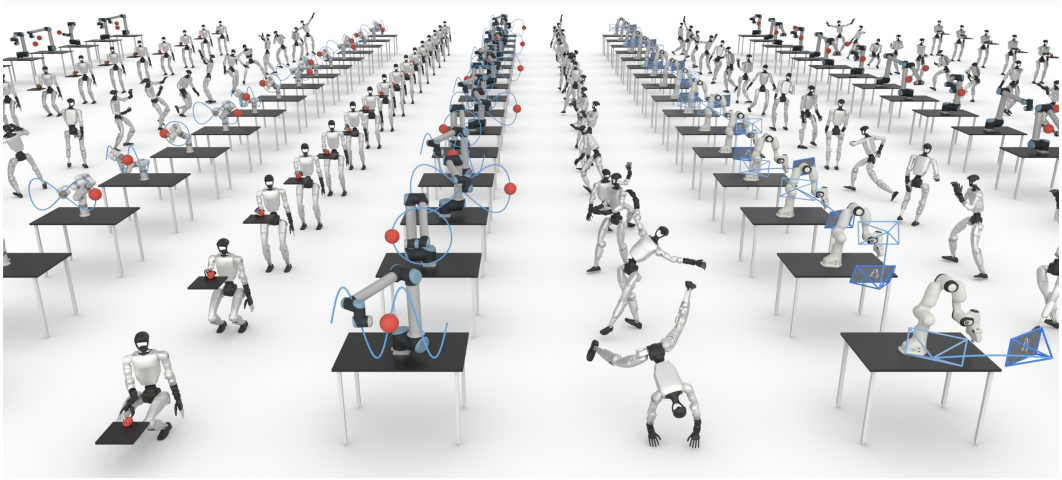


Figure 1: **RoboKit** is a high-performance, general-purpose optimization toolkit for diverse robot tasks. We showcase representative examples; in the first row, from left to right: whole-body grasping, trajectory optimization, whole-body retargeting, and rendering-based calibration.

**Abstract:** We present RoboKit, a high-performance, general-purpose toolkit for parallel differentiable optimization in robotics, built on three principles: (i) *Performance and flexibility*: GPU-parallel multi-seed solvers pair sampling with versatile optimizers, running 2–60× faster than task-specific baselines across seven tasks; (ii) *Geometry, rendering and physics*: hybrid and coarse-to-fine representations accelerate collision, differentiable rendering and simulation, optimizing 4096 grasps in 2.97 s; (iii) *Integration with learning*: costs are differentiable and simulation is parallel, so RoboKit both distills optimization into policies and trains RL; distilled policies run 9–240× faster at comparable accuracy, and RL reaches a GPU baseline's scores up to 2.1× sooner on the same simulator.

**Keywords:** Differentiable Optimization, Robot Kinematics, Policy Distillation

## 1 Introduction

While today's robot learning embraces end-to-end policies trained at scale, classical optimization is quietly everywhere. It is invoked as a low-level skill during teleoperation and deployment [1, 2, 3, 4, 5]; it drives large-scale data generation [6, 7, 8, 9, 10]; it plans through physics in the loop [11, 12]; and its objectives are distilled directly into policies [13, 14, 15, 16]. In this paper, we ask how to better extend the value of optimization, accelerated by modern GPUs, to robot learning.

To this end, we present RoboKit, a performant, general-purpose toolkit supporting diverse tasks, including inverse kinematics (IK), trajectory optimization, motion retargeting, grasp synthesis, rendering-based calibration, and physics-based control. Three principles shape its design: (i) **Performance and flexibility**: it should be fast enough to keep pace with scaling, yet flexible enough to

*Equal contribution. †Project lead.

open new research directions; (ii) **Geometry, rendering and physics**: it should support geometry-aware tasks, differentiable rendering and simulation, which connect perception and action; and (iii) **Integration with learning**: its objectives and solvers should plug seamlessly into learning frameworks such as PyTorch.

*Performance and flexibility.* RoboKit implements a classical optimization framework in Warp [17], pairing multi-seed sampling with diverse optimizers: first-order, quasi-Newton, Gauss-Newton, and sampling-based methods. With well-tuned engineering, it solves over $10\times$ faster than PyRoKi [4] on pure kinematic tasks. Unlike prior frameworks [5, 4], these optimizers let us pick a suitable solver per task (Gauss-Newton for IK, L-BFGS for trajectory optimization), and a composable library of cost terms supports combined tasks such as geometry-aware retargeting.

*Geometry, rendering and physics.* Tasks beyond pure kinematics demand fine-grained geometry and differentiable rendering. RoboKit makes both first-class and accelerates them with voxel-SDF geometry, optional near-surface mesh refinement, and coarse-to-fine rendering. Using voxel SDFs, it synthesizes $4{,}096$ grasps in $2.97\,\mathrm{s}$ versus $178.45\,\mathrm{s}$ for Dex1B [7] with higher success rate, and it calibrates $57\times$ faster than EasyHEC [18] on its easy split (1.23 vs $70.4\,\mathrm{s}$), converging where EasyHEC fails. These capabilities are absent from prior frameworks. With physics in the loop, its MPPI [19] planner raises SPIDER's [12] ADD-AUC from 51.3 to 65.8 on OakInk while running $5.6\times$ faster, and RL on the same physics reaches mjlab's [20] scores up to $2.1\times$ sooner.

*Integration with learning.* Every cost term is PyTorch-native and differentiable end to end, plugging directly into learning. We further study when a learned policy beats the solver and develop a general distillation framework combining offline and online learning. Policies also train with RL directly on RoboKit's parallel physics.

Our contributions are threefold:

- A high-performance parallel optimization toolkit implemented in Warp, with the freedom to mix solvers and cost terms per task.
- Accelerated geometry, rendering and physics, extending optimization to contact-rich, rendering-based and physics-in-the-loop tasks beyond kinematics.
- Differentiable operators that plug into learning frameworks, with a general distillation framework for embedding optimization into policies.

## 2 Related Work

**IK & Trajectory Optimization.** Inverse kinematics seeks a joint configuration reaching a target end-effector pose. Early closed-form solvers [21, 22] relied on restrictive assumptions, so recent work [23, 24, 25, 4] favors iterative optimization for its flexibility, support for secondary objectives, and collision handling across embodiments. Trajectory optimization [26, 1, 2, 4] extends this over time; since each timestep couples mainly to its neighbors, the resulting sparse problems benefit from sparse linear algebra.

**Motion Retargeting.** Motion retargeting adapts existing motion data to new embodiments and is widely studied for both humanoid whole-body motion and dexterous manipulation. For humanoids, joint- and IK-based methods are common, but morphological differences cause artifacts such as foot sliding and penetration, motivating learning-based [27], physics-augmented [28, 29], and contact-aware [30, 31] formulations. Hand retargeting is more challenging due to dense contacts and highly articulated kinematics; approaches include direct joint mapping [32, 33], keypoint-based IK [34, 35, 36, 37], inverse dynamics [38], and functional pose equivalence [39, 40]. Physics-based simulation and learning from demonstrations improve plausibility [41] but are highly sensitive to retargeting quality, and standardized contact-aware methods remain lacking; SPIDER [12] scales physics-based retargeting with sampling in simulation, while ManipTrans [42] and DexMachina [43] learn RL policies that transfer bimanual demonstrations.

Table 1: **Feature and task coverage** of representative optimization tools. ✓✓ denotes two-level coarse-to-fine support.

| Method | Features | | | Tasks | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Hybrid Geometry | Rendering | Physics | Sim. MPC | IK | Traj. Opt. | Hand Retarget. | Humanoid Retarget. | Grasp Synth. | Render. Calib. |
| pink [25], mink [24] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| TrajOpt [58] | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Dex-Retargeting [1] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Geo-Retargeting [15] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| GMR [2] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| PyRoKi [4] | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| cuRobo [5] | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| DIAL-MPC [11] | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| SPIDER [12] | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| **RoboKit (Ours)** | ✓✓ | ✓✓ | ✓✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Grasp Synthesis.** Dexterous grasp synthesis aims to generate stable grasp configurations for multi-fingered hands. Early approaches rely on analytic grasp metrics and optimization-based sampling with known object models [44, 45], but scale poorly to high-DoF hands. Learning-based methods have since become dominant: imitation learning trains on large-scale datasets from human demonstrations [46, 47], force-closure optimization [48, 49], or differentiable simulation [6], enabling scalable supervision [50, 49, 51], while reinforcement learning learns closed-loop policies that are robust across objects [52, 53, 54, 55]. BODex [56] scales optimization-based synthesis with GPU-parallel bilevel optimization. Still, scalable, human-like, and cross-embodiment grasp generation remains an open challenge [57].

**Optimization Libraries.** Existing toolkits each cover only part of the design space. CPU IK solvers such as TracIK [23], pink [25], and mink [24] stay IK-centered; TrajOpt [58] adds collision-aware motion generation; MoveIt [59], Drake [60] and Crocoddyl [61] offer general planning and optimal-control stacks on CPU, and VAMP [62] vectorizes sampling-based planning; and GPU motion planners like cuRobo [5, 63] offer higher throughput. However, such motion-planning libraries typically target collision avoidance rather than contact-rich tasks like grasping. Sampling-based MPC frameworks such as MJPC [64] and DIAL-MPC [11] optimize through physics, but target control alone. Retargeting tools [1, 26] target hands or humanoids, while PyRoKi [4] unifies these tasks into a single system across diverse robots, though it remains limited to kinematics-oriented tasks and primitive geometries. In contrast, RoboKit combines a GPU-parallel solver, accelerated geometry, rendering and physics, and PyTorch integration in a single framework, and is the only one to cover all of these tasks (Tab. 1).

## 3 Method

RoboKit is built on three components: (i) a *parallel solver* (Sec. 3.1) that pairs GPU-parallel multi-seed sampling with diverse optimizers (first-order, quasi-Newton, Gauss-Newton, and sampling-based) to produce high-quality solutions at scale; (ii) an *accelerated structure* (Sec. 3.2) that makes geometry, differentiable rendering and physics first-class via hybrid and coarse-to-fine representations; and (iii) *policy distillation* (Sec. 3.3) that distills optimization into fast policies. An overview of our framework is presented in Fig. 2.

### 3.1 Parallel Solver

**Variables.** The core variable is the joint configuration $q \in \mathbb{R}^d$ ($d$ degrees of freedom), indexed over time as $q_t$ for trajectories. We support fixed, revolute, prismatic, and mimic joints, and $\mathrm{SE}(3)/\mathrm{SO}(3)$ Lie-group variables for poses. A differentiable forward kinematics maps $q$ and base pose ${}^{w}\mathbf{T}_{b}$ to

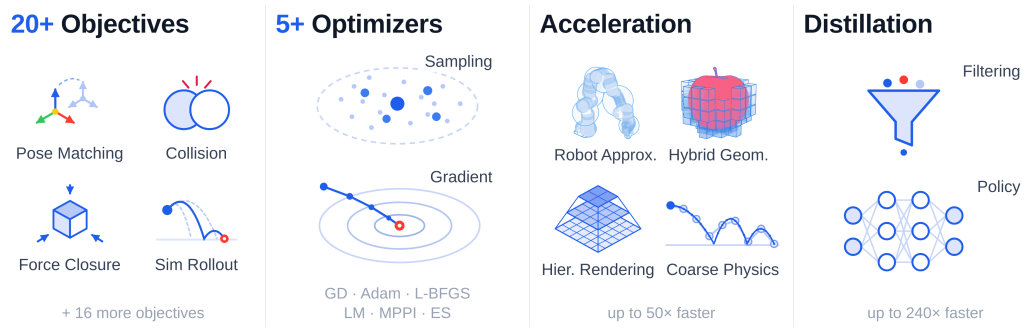



Figure 2: **Method Overview.** RoboKit has three components: a parallel *solver* over differentiable objectives, an *acceleration* structure for geometry, rendering and physics, and *policy distillation* via offline and online learning.

joint poses ${}^{w}\mathbf{T}_i = \mathrm{FK}(q)_i$, with spatial or body Jacobians. We also allow auxiliary variables such as scale factors, common in retargeting to reduce the morphology gap.

**Solver.** Our solver is task-agnostic and implemented in Warp, a GPU programming framework. It pairs (i) a high-level particle-based sampling solver with flexible sampling, gathering, distribution-fitting [65], and resampling schedules, and (ii) a library of low-level optimizers, including first-order (gradient descent, Adam), quasi-Newton (limited-memory BFGS, L-BFGS), Gauss-Newton or Levenberg–Marquardt (LM), and sampling-based methods such as MPPI [19], letting us pick the best solver per task (e.g., Gauss-Newton for IK, L-BFGS for trajectory optimization). Since Warp lacks native sparse Jacobians, we add coordinate-format (COO) sparse Jacobian support for all cost terms to scale to large variable dimensions. Curvature-based steps typically converge within 16–64 iterations; combined with large-scale parallel sampling, this yields a high-performance pipeline (Tab. 2).

**Objectives.** We provide pre-built objective terms covering tasks from inverse kinematics to grasp synthesis, and support user-defined costs; analytic terms include analytical Jacobians, and the solver minimizes the sum of squared residuals. Terms can be combined, weighted, and reused, and also serve as supervision for online distillation. For example, the pose-matching cost penalizes the relative pose error $c_{\text{pose}} = \log\left({}^{w}\mathbf{T}_i^{-1}\,{}^{w}\mathbf{T}_{\text{tgt}}\right) \in \mathfrak{se}(3)$ via the logarithm map; see the supplementary for the full list. Objectives can also score physics rollouts in MuJoCo Warp [66], which sampling-based optimizers minimize without gradients.

### 3.2 Accelerated Structure

Geometry, rendering and physics dominate the per-iteration cost of contact-rich, calibration and simulation tasks. We accelerate them with four structures that exploit spatial locality and a coarse-to-fine schedule.

**Hybrid Voxel SDF and Mesh.** Querying a mesh SDF traverses a BVH and is expensive across thousands of sphere centers per iteration. A precomputed voxel SDF is cheap but its gradient is biased near the surface, where contact reasoning is most sensitive. We therefore pair each object $\mathcal{O}$ with a baked voxel SDF $\mathcal{V}$ (a grid padded by $\delta$ beyond its bounding box) and its source mesh $\mathcal{M}$, reading the cheap voxel field and refining to the exact mesh only inside a near-surface band:

$$\mathrm{SDF}({}^{w}c_i, \mathcal{O}) = \begin{cases} \mathrm{SDF}({}^{w}c_i, \mathcal{M}), & \mathrm{SDF}({}^{w}c_i, \mathcal{V}) \le \tau, \\ \mathrm{SDF}({}^{w}c_i, \mathcal{V}), & \text{otherwise}, \end{cases}$$

with $\tau = bh$, where $h$ is the voxel size and $b$ is the configurable refinement-band width in voxels (default $b = 4$). Far points cost an $O(1)$ voxel sample; near-contact points keep mesh-level accuracy and an exact normal $\hat{n}$.

**Coarse-to-Fine Collision Queries.** The robot is approximated by per-link collision spheres (as in cuRobo [5]). We additionally wrap each link $\ell$ in one coarse bounding sphere (center ${}^{w}c_\ell$, radius $R_\ell$) that contains all of its collision spheres. Mirroring the hybrid query above, the effective distance

$\widehat{\mathrm{SDF}}$ fed to the collision cost runs the exact per-sphere query only when the link is near the scene, and reuses the single broad-phase query otherwise:

$$\widehat{\mathrm{SDF}}({}^{w}c_i, \mathcal{O}) = \begin{cases} \mathrm{SDF}({}^{w}c_i, \mathcal{O}), & \mathrm{SDF}({}^{w}c_\ell, \mathcal{O}) \le R_\ell + \epsilon, \\ \mathrm{SDF}({}^{w}c_\ell, \mathcal{O}), & \text{otherwise.} \end{cases}$$

A link with $\mathrm{SDF}({}^{w}c_\ell, \mathcal{O}) > R_\ell + \epsilon$ keeps all of its spheres beyond the safety margin $\epsilon$, so the cost is inactive and one broad query prunes the whole link, which is especially effective for hands and articulated arms.

**Coarse-to-Fine Rendering.** Rendering-based calibration matches rendered robot masks to observations, a non-convex silhouette objective. We use a resolution pyramid: early stages render at low resolution with many seeds and a strongly blurred, smoother cost with fewer local minima so sampling escapes poor basins cheaply; later stages raise resolution, prune to the best seeds via beam selection, and remove the blur for sub-pixel refinement. A single-level full-resolution solver fails under large initialization error, whereas the pyramid converges to sub-degree accuracy.

**Coarse-to-Fine Physics.** Sampling-based MPC and RL spend most of their time stepping the simulator. Early iterations run a coarse scene model, with larger timesteps, fewer solver iterations and fewer collision pairs; later ones switch to full physics.

### 3.3 Policy Learning

Policy distillation transfers the solver's knowledge to a policy through two complementary paradigms: *offline* and *online* distillation.

**Offline Distillation.** We run the solver over a distribution of task instances to generate solved configurations. Since poorly converged or infeasible solutions degrade learning, we optionally filter to keep only high-quality demonstrations, e.g., by thresholding the optimization cost or verifying physical plausibility in simulation; this is critical for hard tasks like grasping. Given the filtered dataset $\mathcal{D} = \{(o_i, q_i^*)\}$, we train a policy $\pi_\theta(o)$ via supervised learning:

$$\mathcal{L}_{\mathrm{offline}}(\theta) = \mathbb{E}_{(o,q^*)\sim\mathcal{D}} \left[ \left\| \pi_\theta(o) - q^* \right\|^2 \right].$$

**Online Distillation.** Online distillation trains the policy directly from optimization objectives without a fixed dataset, backpropagating gradients to the policy weights:

$$\mathcal{L}_{\mathrm{online}}(\theta) = \mathbb{E}_{o\sim p(o)} \left[ \frac{1}{2} \sum_k \left\| W_k \, c_k\big(\pi_\theta(o); o\big) \right\|_2^2 \right],$$

where $c_k(q; o)$ is the $k$-th unweighted residual vector and $W_k$ is its diagonal residual-weight matrix, including the task gain. Gradients flow through these differentiable residuals to the policy; the solver is never unrolled or differentiated. Operating on-policy, it provides exact gradients and draws fresh samples each iteration, typically outperforming offline distillation and generalizing better.

**Integrating Offline and Online.** Online distillation performs better but depends on task complexity. For simple tasks like position-only retargeting it suffices alone; for contact-rich tasks like grasp synthesis, pure online training converges to poor local minima, so we warm up with offline supervised learning on filtered demonstrations before online fine-tuning. This hybrid recipe mirrors imitation and reinforcement learning practice, but benefits from the stable gradients of differentiable kinematic objectives.

## 4 Experiments

We address three questions in experiments: (1) How well does our optimization framework perform? (2) How effective are policies learned with RoboKit? (3) Which design choices matter? We answer these questions below.

Table 2: **Quantitative comparison** of RoboKit against baselines across seven tasks. See the appendix for metric definitions.

(a) Batched IK on the 7-DoF Franka Panda.

| | PyRoKi [4] | | | | Ours (opt) | | | |
|---|---|---|---|---|---|---|---|---|
| Batch | Time (ms) | Succ. (%) | Pos. Err (mm) | Ori. Err (rad) | Time (ms) | Succ. (%) | Pos. Err (mm) | Ori. Err (rad) |
| 1 | 9.08 | 100.00 | $1.74 \times 10^{-4}$ | $1.88 \times 10^{-7}$ | **0.65** | 100.00 | $1.23 \times 10^{-4}$ | $1.68 \times 10^{-7}$ |
| 10 | 14.52 | 100.00 | $3.80 \times 10^{-2}$ | $4.86 \times 10^{-5}$ | **0.61** | 100.00 | $1.15 \times 10^{-2}$ | $1.90 \times 10^{-5}$ |
| 100 | 16.04 | 100.00 | $1.91 \times 10^{-2}$ | $2.13 \times 10^{-5}$ | **0.81** | 100.00 | $1.08 \times 10^{-3}$ | $1.20 \times 10^{-6}$ |
| 1000 | 48.19 | 99.88 | $1.47 \times 10^{-3}$ | $1.01 \times 10^{-6}$ | **3.41** | 99.94 | $4.04 \times 10^{-4}$ | $4.15 \times 10^{-7}$ |
| 2000 | 86.20 | 99.89 | $1.57 \times 10^{-3}$ | $1.05 \times 10^{-6}$ | **7.12** | 99.91 | $3.28 \times 10^{-4}$ | $4.03 \times 10^{-7}$ |

(b) Motion Planning for the Franka arm.

| Method | Success (%) ↑ | Plan Mean (ms) ↓ | Plan P98 (ms) ↓ | Motion Time (s) ↓ | Path Length (rad) ↓ | Max Jerk (rad/s$^3$) ↓ | Mean Vel (rad/s) | Max Acc (rad/s$^2$) ↓ |
|---|---|---|---|---|---|---|---|---|
| cuRobo [5] | 99.57 | 56.6 | 214.5 | **1.181** | 3.145 | 146.2 | 0.698 | **11.567** |
| Ours (opt) | **99.65** | **16.3** | **40.5** | 1.189 | **3.131** | **118.1** | 0.766 | 11.941 |

(c) Online hand retargeting for Shadow Hand.

| Method | Time (ms) ↓ | Pos. (mm) ↓ | Pinch (%) ↑ | Open (%) ↑ | Acc. (rad/s$^2$) ↓ |
|---|---|---|---|---|---|
| Dex-Ret. (pos) [1] | 3.66 | 28.13 | 37.5 | 65.2 | 7.44 |
| Dex-Ret. (DexPilot) [1] | 15.71 | 15.69 | 96.9 | 85.2 | **4.23** |
| PyRoKi [4] | 33.19 | 10.76 | 35.7 | **94.3** | 31.02 |
| Ours (opt) | **1.25** | **6.05** | **99.7** | 92.3 | 5.82 |

(d) Online humanoid retargeting for G1.

| Method | Time (ms) ↓ | MPJPE (cm) ↓ | WA-MPJPE (cm) ↓ | Acc. (rad/s$^2$) ↓ |
|---|---|---|---|---|
| GMR [2] | 7.91 | 9.26 | 9.21 | 20.06 |
| Ours (opt) | **2.00** | **8.08** | **5.57** | **16.98** |

(e) Grasp synthesis.

| Method | Time (s) ↓ | SR (%) ↑ | Q1 ↑ | Pen. (mm) ↓ |
|---|---|---|---|---|
| Dex1B [7] | 178.45 | 23.63 | **0.109** | 1.06 |
| Ours (opt) | **2.97** | **46.83** | 0.077 | **0.70** |

(f) Hand-eye calibration on Panda (realistic).

| Method | Time (s) ↓ | Trans. (cm) ↓ | Rot. (°) ↓ |
|---|---|---|---|
| EasyHEC [18] | 458.5 | 96.43 | 12.66 |
| Ours (opt) | **2.0** | **1.10** | **0.54** |

(g) Physics-based dexterous retargeting on five hands (SPIDER's MuJoCo comparison); time per frame.

| | OakInk (143 trials) | | | | | GigaHands (75 trials) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | Time (ms) ↓ | ADD-AUC (%) ↑ | Succ. (%) ↑ | Pos. (cm) ↓ | Rot. (°) ↓ | Time (ms) ↓ | ADD-AUC (%) ↑ | Succ. (%) ↑ | Pos. (cm) ↓ | Rot. (°) ↓ |
| SPIDER [12] | 610 | 51.3 | 59 | 4.38 | 34.8 | 125 | 50.3 | 77 | **4.05** | 16.0 |
| Ours (opt) | **109** | **65.8** | **86** | **3.17** | **15.2** | **57** | **63.7** | **84** | 4.11 | **9.5** |

### 4.1 Experimental Settings

We evaluate RoboKit on six kinematic tasks and one physics-based task, reusing established benchmarks: the cuRobo [5] benchmark for inverse kinematics and trajectory optimization (7-DoF Franka Panda), Dex-Retargeting [1] data for hand motion retargeting (Shadow Hand), AMASS [67] for human motion retargeting (Unitree G1), Dex1B [7] for grasp synthesis (MANO on DexGraspNet [6]), the EasyHEC [18] benchmark for rendering-based calibration (Panda), and SPIDER's [12] MuJoCo comparison for physics-based dexterous retargeting (five hands on OakInk and GigaHands), and mjlab's [20] RL tasks for policy learning (Go1, G1 and a Yam arm), a GPU framework in the line of Isaac Gym [68] and MuJoCo Playground [69]. We report solve time alongside the task-specific success and accuracy metrics defined in each table. All benchmark comparisons (Tab. 2) run on a single NVIDIA RTX 4090 GPU; timings are averaged over 5 runs (3 for human retargeting, one pass over all frames for hand retargeting) after a warmup.

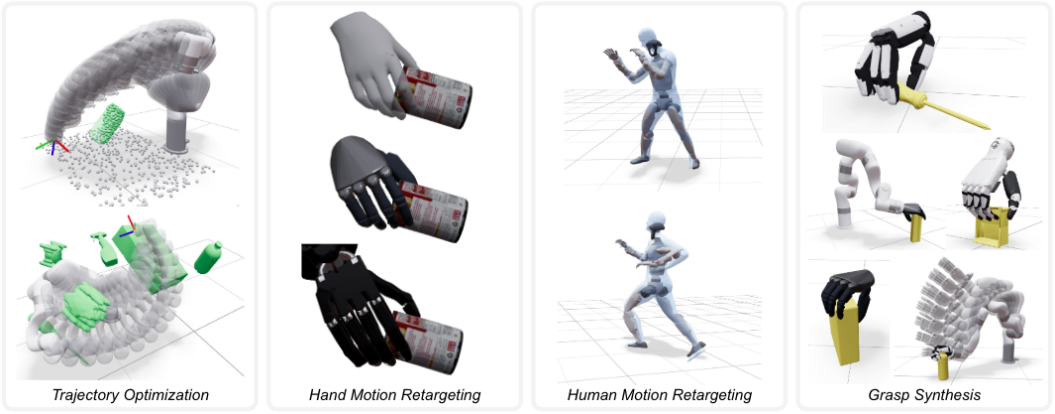



Figure 3: **Qualitative results** of the learned policies on trajectory optimization, hand and full-body retargeting, and dexterous grasp synthesis.

### 4.2 Parallel Solver Performance

We compare RoboKit against optimization baselines across all seven tasks (Tab. 2); it is consistently the fastest, with task-specific tradeoffs in accuracy and solution quality.

*(a) Inverse Kinematics.* Against PyRoKi [4] both methods use Levenberg–Marquardt, 64 initial seeds refined to 4, and a 16-iteration budget. The resulting speedup (e.g., 7.1 vs 86.2 ms at batch 2000) reflects implementation and solver-setting differences: our Warp [17] backend removes the per-iteration framework overhead of JAX, especially at large batches, while reaching the same near-100% success and comparable accuracy.

*(b) Trajectory Optimization.* We adopt the strong cuRobo [5] benchmark and, like it, optimize with L-BFGS. Both methods are judged on goal reach, world and self-collision, dynamic limits, and joint limits with the benchmark's 0.2 rad tolerance. RoboKit is faster (0.016 vs 0.057 s plan time, and a far lower 98th-percentile time), maintains a comparable success rate, and yields a shorter joint path and lower jerk, at the cost of slightly longer motion time. We omit PyRoKi here: its LM solver is prohibitively slow in this setting, and it performs trajectory optimization without time reparameterization, so its motion-level metrics are not directly comparable. Note that the obstacles here are primitive boxes, so our hybrid geometry acceleration (Sec. 3.2) is not exercised; the gains come purely from the solver and engineering.

*(c) Hand Retargeting.* Dex-Retargeting [1] optimizes a single objective at a time (position *or* vector), whereas RoboKit and PyRoKi jointly satisfy both. Even against these single-objective specialists, RoboKit attains lower position error, and is 26.6× faster than PyRoKi at a higher pinch success rate.

*(d) Human Retargeting.* Against GMR [2], we retarget human motion from the AMASS dataset [67] to the G1 humanoid; RoboKit cuts the per-frame time by about 4× (2.00 vs 7.91 ms) while lowering both MPJPE and WA-MPJPE.

*(e) Grasp Synthesis.* RoboKit optimizes 4,096 grasps using voxel-SDF geometry in 2.97 s, versus 178.45 s for Dex1B [7] (60× faster). Optimization uses a precomputed voxel field; all grasps are scored using the object mesh. It has higher success (46.83% vs 23.63%) and lower penetration (0.70 vs 1.06 mm), while Dex1B achieves higher Q1 (0.109 vs 0.077). Mesh results and the geometry ablation appear in Tab. 11 and Tab. 5d.

*(f) Rendering-Based Calibration.* We evaluate the realistic setting matching our introduction (35 cm/45° initial error with sensor and calibration noise). The coarse-to-fine pyramid (Sec. 3.2) reaches 1.10 cm in 2.0 s, while EasyHEC [18] does not converge (96.43 cm). On the easy split (10 cm/15° initial error, no added noise), where both reach under 0.3 cm, we are 57× faster (1.23 s vs 70.4 s).

Table 3: **Distilled policy performance** across four tasks.

(a) Trajectory optimization.

| Method | Time (ms) ↓ | SR (%) ↑ | Pos. (cm) | Rot. (°) | Smooth. |
|---|---|---|---|---|---|
| Ours (opt) | 150.61 | **96.72** | **1.04** | **1.49** | **0.256** |
| Ours (distilled) | **0.63** | 95.90 | 1.08 | 2.32 | 0.588 |

(b) Hand retargeting.

| Method | Time (ms) ↓ | SR (%) ↑ | Pos. (mm) ↓ | Vec. (mm) ↓ |
|---|---|---|---|---|
| Ours (opt) | 5.22 | 99.72 | 7.79 | 8.81 |
| Ours (distilled) | **0.44** | **100.0** | **4.80** | **6.5** |

(c) Humanoid retargeting.

| Method | Time (ms) ↓ | Pos. (cm) ↓ | Ori. (rad) ↓ |
|---|---|---|---|
| Ours (opt) | 10.86 | **7.15** | **0.1613** |
| Ours (distilled) | **0.72** | 9.92 | 0.1936 |

(d) Grasp synthesis.

| Method | Time (ms) ↓ | SR (%) ↑ | Q1 ↑ | Pen. (cm) ↓ |
|---|---|---|---|---|
| UGG [73] | – | 43.6 | 0.026 | 0.43 |
| Dex1B [7] | 1.71 | 63.7 | 0.075 | 0.29 |
| Ours (opt) | 14.87 | 31.25 | **0.110** | 0.99 |
| Ours (distilled) | **1.71** | **71.3** | 0.089 | **0.18** |

Table 4: **RL training** against mjlab [20] on the same MuJoCo Warp physics: minutes to reach 95% / 100% of mjlab's final score, median over seeds.

| Method | Go1 Flat | G1 Flat | Go1 Rough | G1 Rough | G1 Tracking | Yam Lift |
|---|---|---|---|---|---|---|
| mjlab [20] | 7.6 / 37.6 | 18.5 / 47.2 | 20.0 / 45.6 | 77.6 / 102.1 | 49.0 / 51.1 | 7.4 / 16.8 |
| Ours | **3.2 / 17.7** | **9.2 / 26.7** | **12.7 / 34.4** | **46.4 / 74.6** | **21.9 / 29.3** | **3.8 / 8.7** |

*(g) Physics-Based Retargeting.* Following SPIDER [12], we retarget human hand–object motion to five dexterous hands by sampling-based MPC in MuJoCo Warp [70, 66], on 7 OakInk [71] tasks and 3 GigaHands [72] clips with five seeds each; only the optimizer differs from SPIDER. RoboKit raises ADD-AUC from 51.3 to 65.8 on OakInk and from 50.3 to 63.7 on GigaHands, and is 5.6× and 2.2× faster.

### 4.3 Policy Learning Performance

We evaluate policies distilled across four tasks, excluding inverse kinematics, which optimization already solves trivially; results are in Tab. 3 (distilled rows in blue) and Fig. 3. Overall, distilled policies run 9–240× faster than optimization at competitive accuracy, enabling real-time deployment. All methods use offline training.

Policies vs. our optimizer, and vs. learned grasping baselines (UGG, Dex1B). Trajectory and human-retargeting times are per frame. Optimizer rows follow each policy's evaluation protocol on an RTX A5000, which differs from Tab. 2.

Per-task results follow. *(a) Trajectory optimization:* the policy amortizes the solver into a 239× faster forward pass (0.63 vs. 150.61 ms/frame) at matched success, with only a slight smoothness drop. *(b) Hand retargeting:* sub-millisecond inference that also *improves* accuracy over the optimizer (position error 7.79 → 4.80 mm), as in-distribution training escapes the local minima the optimizer settles into. *(c) Human retargeting:* a 15× speedup (0.72 vs. 10.86 ms/frame) with only a modest accuracy drop. *(d) Grasp synthesis:* our distilled policy achieves higher success and lower penetration than UGG [73], Dex1B [7], and the optimizer, with competitive grasp quality. Beyond distillation, RoboKit trains RL policies on mjlab's tasks with the same physics (Tab. 4), reaching mjlab's final score 1.3–2.1× sooner.

### 4.4 Ablation Study

**Offline vs. Online Distillation.** Online (Supervised) learns fresh RoboKit-optimized targets, whereas Online (Direct Grad.) directly minimizes the task objective. On the simpler hand-retargeting task, pure online distillation wins, as on-policy sampling yields more accurate gradients (Tab. 5a); here we sample random hand configurations rather than DexYCB motions so the gap is

Table 5: **Ablation study** of distillation, demonstration filtering, and geometry.

(a) Offline vs. online (hand retargeting).

| Method | SR (%) ↑ | Pos. Err (mm) ↓ | Vec. Err (mm) ↓ |
|---|---|---|---|
| Offline | 100.0 | 4.71 | 7.86 |
| Online (Supervised) | 100.0 | 4.34 | 6.87 |
| Online (Direct Grad.) | 100.0 | **2.71** | **6.59** |

(b) Offline vs. hybrid (grasp).

| Method | SR (%) ↑ | Q1 ↑ | Pen. Err (mm) ↓ |
|---|---|---|---|
| Offline | 57.91 | 0.118 | 3.51 |
| Offline + Online | **65.48** | **0.126** | **2.13** |

(c) Demonstration filtering (grasp).

| Method | SR (%) ↑ | Q1 ↑ | Pen. Err (mm) ↓ |
|---|---|---|---|
| Offline (w/o. filtering) | 23.83 | 0.079 | 6.82 |
| Offline | **57.91** | **0.118** | **3.51** |

(d) Geometry (grasp): voxel-only, scored on the mesh.

| Geometry | Time (s) ↓ | SR (%) ↑ | Pen. (mm) ↓ |
|---|---|---|---|
| Mesh SDF (BVH) | 16.61 | **48.49** | **0.55** |
| Voxel SDF (Ours) | **2.97** | 46.83 | 0.70 |

visible. On the harder grasp task, pure online training diverges in the non-convex landscape, so we warm-start offline and fine-tune online; this hybrid recipe yields the best success, quality, and penetration (Tab. 5b).

**Effect of Demonstration Filtering.** Training offline on unfiltered solver outputs underperforms even the optimizer; simulation-based filtering to keep only successful grasps roughly doubles success and nearly halves penetration (Tab. 5c), confirming that data quality is critical for contact-rich tasks.

**Geometry Representation.** Holding the solver fixed, replacing exact mesh-BVH queries with a precomputed voxel SDF cuts grasp-optimization time about $5.6\times$ at comparable success and negligible penetration change (Tab. 5d).

**Solver Components.** Table 6 isolates the solver's components on Franka IK. PyTorch kernels slow solves $77\times$ ($1.82 \rightarrow 139.59$ ms) and numerical Jacobians $10\times$, both at $100\%$ success; removing multi-seed sampling barely saves time but drops success to $72.6\%$.

Table 6: **Solver component ablation** on the 7-DoF Franka Panda.

| Variant | Time (ms) ↓ | SR (%) ↑ |
|---|---|---|
| Full | **1.82** | **100.0** |
| w/o Warp | 139.59 | 100.0 |
| w/o Anal. Jac | 17.93 | 100.0 |
| w/o Particle | 1.64 | 72.6 |

## 5 Conclusion

We presented RoboKit, a high-performance, general-purpose toolkit for parallel differentiable optimization in robotics. Built in Warp with versatile optimizers, it runs 2–$60\times$ faster than task-specific baselines; it makes geometry, differentiable rendering and physics first-class; and its differentiable costs and parallel simulation plug into learning, distilling optimization into policies that run 9–$240\times$ faster at comparable accuracy. We hope RoboKit offers a practical foundation for real-time manipulation, teleoperation, and learning-from-demonstration.

## 6 Limitations

RoboKit targets GPU-parallel workloads: for single-instance kinematics on CPU, kernel-launch overhead dominates and a C++ solver would be faster. Its trajectory optimization has no graph-search stage, which current benchmarks do not stress, and it lacks capabilities such as inverse dynamics.

## Acknowledgment

This work was supported, in part, by NSF CAREER Award IIS-2240014 and NSF CCF-2112665 (TILOS).

## References

[1] Y. Qin, W. Yang, B. Huang, K. Van Wyk, H. Su, X. Wang, Y.-W. Chao, and D. Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. In *RSS*, 2023.

[2] J. P. Araujo, Y. Ze, P. Xu, J. Wu, and C. K. Liu. Retargeting matters: General motion retargeting for humanoid motion tracking. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

[3] M. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, G. Wang, F.-F. Li, G. Shi, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

[4] C. M. Kim, B. Yi, H. Choi, Y. Ma, K. Goldberg, and A. Kanazawa. Pyroki: A modular toolkit for robot kinematic optimization. In *IROS*, 2025.

[5] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. R. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. D. Ratliff, and D. Fox. Curobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 8112–8119. IEEE, 2023. doi:10.1109/ICRA48891.2023.10160765. URL https://doi.org/10.1109/ICRA48891.2023.10160765.

[6] R. Wang, J. Zhang, J. Chen, Y. Xu, P. Li, T. Liu, and H. Wang. Dexgraspnet: A large-scale robotic dexterous grasp dataset for general objects based on simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11359–11366. IEEE, 2023.

[7] J. Ye, K. Wang, C. Yuan, R. Yang, Y. Li, J. Zhu, Y. Qin, X. Zou, and X. Wang. Dex1b: Learning with 1b demonstrations for dexterous manipulation. In *RSS*, 2025.

[8] J. Yang, J. J. Liu, Y. Li, Y. Khaky, K. Shaw, and D. Pathak. Deep reactive policy: Learning reactive manipulator motion planning for dynamic environments. *arXiv preprint arXiv:2509.06953*, 2025.

[9] M. Dalal, J. Yang, R. Mendonca, Y. Khaky, R. Salakhutdinov, and D. Pathak. Neural mp: A generalist neural motion planner. *arXiv preprint arXiv:2409.05864*, 2024.

[10] L. Yang, X. Huang, Z. Wu, A. Kanazawa, P. Abbeel, C. Sferrazza, C. K. Liu, R. Duan, and G. Shi. Omniretarget: Interaction-preserving data generation for humanoid whole-body loco-manipulation and scene interaction. *arXiv preprint arXiv:2509.26633*, 2025.

[11] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi. Full-order sampling-based MPC for torque-level locomotion control via diffusion-style annealing. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[12] C. Pan, C. Wang, H. Qi, J. Urain, Z. Liu, H. Bharadhwaj, A. Sharma, T. Wu, G. Shi, J. Malik, and F. Hogan. Spider: Scalable physics-informed dexterous retargeting. *arXiv preprint arXiv:2511.09484*, 2025.

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

[14] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam. Theseus: A Library for Differentiable Nonlinear Optimization. *Advances in Neural Information Processing Systems*, 2022.

[15] Z.-H. Yin, C. Wang, L. Pineda, K. Bodduluri, T. Wu, P. Abbeel, and M. Mukadam. Geometric retargeting: A principled, ultrafast neural hand retargeting algorithm. *arXiv preprint arXiv:2503.07541*, 2025.

[16] R. Zurbrügg, A. Cramariuc, and M. Hutter. Graspqp: Differentiable optimization of force closure for diverse and robust dexterous grasping. *arXiv preprint arXiv:2508.15002*, 2025.

[17] M. Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, 2022. NVIDIA GPU Technology Conference (GTC).

[18] L. Chen, Y. Qin, X. Zhou, and H. Su. Easyhec: Accurate and automatic hand-eye calibration via differentiable rendering and space exploration. *IEEE Robotics and Automation Letters (RA-L)*, 2023.

[19] G. Williams, A. Aldrich, and E. A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[20] K. Zakka, Q. Liao, B. Yi, L. Le Lay, K. Sreenath, and P. Abbeel. mjlab: A lightweight framework for GPU-accelerated robot learning. *arXiv preprint arXiv:2601.22074*, 2026.

[21] R. Diankov. *Automated construction of robotic manipulation programs*. PhD thesis, Carnegie Mellon University, USA, 2010.

[22] D. L. Pieper. *The kinematics of manipulators under computer control*. PhD thesis, Stanford University, 1968.

[23] P. Beeson and B. Ames. TRAC-IK: an open-source library for improved solving of generic inverse kinematics. In *15th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2015, Seoul, South Korea, November 3-5, 2015*, pages 928–935. IEEE, 2015. doi:10.1109/HUMANOIDS.2015.7363472. URL https://doi.org/10.1109/HUMANOIDS.2015.7363472.

[24] K. Zakka. Mink: Python inverse kinematics based on MuJoCo, Dec. 2025. URL https://github.com/kevinzakka/mink.

[25] S. Caron, Y. De Mont-Marin, R. Budhiraja, S. H. Bang, I. Domrachev, S. Nedelchev, P. Du, A. Escande, J. Vaillant, B. Wingo, and S. Patapati. Pink: Python inverse kinematics based on Pinocchio, 2026. URL https://github.com/stephane-caron/pink.

[26] T. He, Z. Luo, W. Xiao, C. Zhang, K. Kitani, C. Liu, and G. Shi. Learning human-to-humanoid real-time whole-body teleoperation. *arXiv preprint arXiv:2403.04436*, 2024.

[27] K. Aberman, P. Li, D. Lischinski, O. Sorkine-Hornung, D. Cohen-Or, and B. Chen. Skeleton-aware networks for deep motion retargeting. *ACM Transactions on Graphics (TOG)*, 39(4): 62–1, 2020.

[28] J. Won and J. Lee. Learning body shape variation in physics-based characters. *ACM Transactions on Graphics*, 38(6), 2019.

[29] H. Ryu, M. Kim, S. Lee, M. S. Park, K. Lee, and J. Lee. Functionality-driven musculature retargeting. *Computer Graphics Forum*, 40(1):341–356, 2021.

[30] R. Villegas, D. Ceylan, A. Hertzmann, J. Yang, and J. Saito. Contact-aware retargeting of skinned motion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9720–9729, 2021.

[31] Y. Zhang, D. Gopinath, Y. Ye, J. Hodgins, G. Turk, and J. Won. Simulation and retargeting of complex multi-character interactions. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023.

[32] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[33] V. Kumar and E. Todorov. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 657–663, 2015.

[34] B. Humberston and D. K. Pai. Hands on: Interactive animation of precision manipulation and contact. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, page 63–72, 2015.

[35] Y. Qin, H. Su, and X. Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *IEEE Robotics and Automation Letters*, 7: 10873–10881, 2022.

[36] D. Antotsiou, G. Garcia-Hernando, and K. T. K. Task-oriented hand motion retargeting for dexterous manipulation imitation. In *The European Conference on Computer Vision (ECCV) Workshops*, 2018.

[37] S. Dasari, A. Gupta, and V. Kumar. Learning dexterous manipulation from exemplar object trajectories and pre-grasps. In *IEEE International Conference on Robotics and Automation*, 2023.

[38] P. G. Kry and D. K. Pai. Interaction capture and synthesis. *ACM Transactions on Graphics*, 25 (3):872–880, 2006.

[39] A. Handa, K. V. Wyk, W. Yang, J. Liang, Y. W. Chao, Q. Wan, S. Birchfield, N. D. Ratliff, and D. Fox. Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9164–9170, 2019.

[40] A. Sivakumar, K. Shaw, and D. Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube. In *Proceedings of Robotics: Science and Systems (RSS)*, 2022.

[41] Y. H. Wu, J. Wang, and X. Wang. Learning generalizable dexterous manipulation from human grasp affordance. In *Conference on Robot Learning (CORL)*, 2022.

[42] K. Li, P. Li, T. Liu, Y. Li, and S. Huang. ManipTrans: Efficient dexterous bimanual manipulation transfer via residual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[43] Z. Mandi, Y. Hou, D. Fox, Y. Narang, A. Mandlekar, and S. Song. DexMachina: Functional retargeting for bimanual dexterous manipulation. *arXiv preprint arXiv:2505.24853*, 2025.

[44] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. In *IEEE Robotics & Automation Magazine*, 2004.

[45] T. Zhu, R. Wu, X. Lin, and Y. Sun. Toward human-like grasp: Dexterous grasping via semantic representation of object-hand. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[46] O. Taheri, N. Ghorbani, M. J. Black, and D. Tzionas. Grab: A dataset of whole-body human grasping of objects. In *European conference on computer vision*, pages 581–600. Springer, 2020.

[47] Y. Liu, Y. Yang, Y. Wang, X. Wu, J. Wang, Y. Yao, S. Schwertfeger, S. Yang, W. Wang, J. Yu, et al. Realdex: towards human-like grasping for robotic dexterous hand. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6859–6867, 2024.

[48] T. Liu, Z. Liu, Z. Jiao, Y. Zhu, and S.-C. Zhu. Synthesizing diverse and physically stable grasps with arbitrary hand structures using differentiable force closure estimator. *IEEE Robotics and Automation Letters*, 7(1):470–477, 2021.

[49] P. Li, T. Liu, Y. Li, Y. Geng, Y. Zhu, Y. Yang, and S. Huang. Gendexgrasp: Generalizable dexterous grasping. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8068–8074. IEEE, 2023.

[50] L. Shao, F. Ferreira, M. Jorda, V. Nambiar, J. Luo, E. Solowjow, J. A. Ojea, O. Khatib, and J. Bohg. Unigrasp: Learning a unified model to grasp with multifingered robotic hands. *IEEE Robotics and Automation Letters*, 5(2):2286–2293, 2020.

[51] S. Liu, Y. Zhou, J. Yang, S. Gupta, and S. Wang. Contactgen: Generative contact modeling for grasp generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20609–20620, 2023.

[52] Y. Xu, W. Wan, J. Zhang, H. Liu, Z. Shan, H. Shen, R. Wang, H. Geng, Y. Weng, J. Chen, et al. Unidexgrasp: Universal robotic dexterous grasping via learning diverse proposal generation and goal-conditioned policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4737–4746, 2023.

[53] W. Wan, H. Geng, Y. Liu, Z. Shan, Y. Yang, L. Yi, and H. Wang. Unidexgrasp++: Improving dexterous grasping policy learning via geometry-aware curriculum and iterative generalist-specialist learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3891–3902, 2023.

[54] T. Wu, M. Wu, J. Zhang, Y. Gan, and H. Dong. Learning score-based grasping primitive for human-assisting dexterous grasping. *Advances in Neural Information Processing Systems*, 2024.

[55] Y. Qin, B. Huang, Z.-H. Yin, H. Su, and X. Wang. Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning*, 2023.

[56] J. Chen, Y. Ke, and H. Wang. Bodex: Scalable and efficient robotic dexterous grasp synthesis using bilevel optimization. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 01–08. IEEE, 2025.

[57] H. Li, W. Mao, W. Deng, C. Meng, H. Fan, T. Wang, Y. Osamu, P. Tan, H. Wang, and X. Deng. Multi-grasplm: A multimodal llm for multi-hand semantic guided grasp generation. *arXiv preprint arXiv:2412.08468*, 2024.

[58] J. Schulman, Y. Duan, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014. doi:10.1177/0278364914528132. URL https://doi.org/10.1177/0278364914528132.

[59] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE robotics & automation magazine*, 19(1):18–19, 2012.

[60] R. Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics. https://drake.mit.edu, 2019.

[61] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[62] W. Thomason, Z. Kingston, and L. E. Kavraki. Motions in microseconds via vectorized sampling-based planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[63] B. Sundaralingam, A. Murali, and S. Birchfield. curobov2: Dynamics-aware motion generation with depth-fused distance fields for high-dof robots. *arXiv preprint arXiv:2603.05493*, 2026.

[64] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa. Predictive sampling: Real-time behaviour synthesis with MuJoCo. *arXiv preprint arXiv:2212.00541*, 2022.

[65] N. Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[66] Google DeepMind and NVIDIA. MuJoCo Warp: GPU-optimized version of the MuJoCo physics simulator. `https://github.com/google-deepmind/mujoco_warp`, 2025.

[67] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black. Amass: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5442–5451, 2019.

[68] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance GPU based physics simulation for robot learning. In *NeurIPS Datasets and Benchmarks Track*, 2021.

[69] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, C. Sferrazza, Y. Tassa, and P. Abbeel. MuJoCo Playground. *arXiv preprint arXiv:2502.08844*, 2025.

[70] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[71] X. Zhan, L. Yang, Y. Zhao, K. Mao, H. Xu, Z. Lin, K. Li, and C. Lu. OAKINK2: A dataset of bimanual hands-object manipulation in complex task completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[72] R. Fu, D. Zhang, A. Jiang, W. Fu, A. Funk, D. Ritchie, and S. Sridhar. GigaHands: A massive annotated dataset of bimanual hand activities. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2025.

[73] J. Lu, H. Kang, H. Li, B. Liu, Y. Yang, Q. Huang, and G. Hua. Ugg: Unified generative grasping. In *ECCV*, 2024.

[74] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. MotionBenchMaker: A tool to generate and benchmark motion planning datasets. *IEEE Robotics and Automation Letters*, 7(2):882–889, 2022.

[75] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox. Motion policy networks. In *Conference on Robot Learning (CoRL)*, 2022.

[76] Y.-W. Chao, W. Yang, Y. Xiang, P. Molchanov, A. Handa, J. Tremblay, Y. S. Narang, K. Van Wyk, U. Iqbal, S. Birchfield, et al. Dexycb: A benchmark for capturing hand grasping of objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9044–9053, 2021.

[77] S. Shin, J. Kim, E. Halilaj, and M. J. Black. WHAM: Reconstructing world-grounded humans with accurate 3D motion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[78] J. Romero, D. Tzionas, and M. J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 36(6), 2017.

[79] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k. Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[80] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[81] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619. IEEE, 2019.

# Appendix

In this supplementary material, we first detail the experimental settings, including the hardware, per-task definitions and metrics, and baselines. Second, we provide additional experiments that complement the paper. Third, we describe the implementation details, including the list of built-in cost terms and the algorithm details. Finally, we present additional qualitative results. Please see our **video** for an overview of our work and additional visualizations.

## A Experimental Details

### A.1 Hardware and Profiling

Our benchmark platform consists of a single NVIDIA RTX 4090 GPU paired with an Intel Xeon Gold 5120 CPU @ 2.20 GHz. Unless otherwise noted, all reported timings are averaged over 5 runs after warmup, which ensures that GPU kernels are compiled and cached before measurement. Unless stated otherwise, time refers to the total wall-clock solve time for the full batch.

### A.2 Task Definitions

We evaluate RoboKit on six kinematic tasks, one physics-based task, and RL training. For each task we describe the dataset and embodiment, the optimization objective, and the evaluation protocol.

**Inverse Kinematics.** Following the cuRobo [5] benchmarks, we solve IK on a fixed-base 7-DoF Franka Panda arm. The objective combines the pose-matching cost $c_{\text{pose}}$ for end-effector alignment with the joint-limit cost $c_{\text{limit}}$. We report solve time, success rate (position error $< 5$ mm and orientation error $< 0.05$ rad), and 98th-percentile position (mm) and orientation (rad) errors over successful instances across batch sizes from 1 to 2000.

**Trajectory Optimization.** We generate collision-free reaching trajectories for the Franka arm in cluttered scenes from MotionBenchMaker [74] and MPiNets [75], following the cuRobo [5] benchmark. The objective includes the goal pose-matching cost $c_{\text{pose}}$, smoothness cost $c_{\text{smooth}}$, joint-limit cost $c_{\text{limit}}$, and collision-avoidance cost $c_{\text{coll}}$. A trajectory is counted as successful if it reaches the goal (position error $\leq 5$ mm and $\sin(\theta/2) \leq 0.05$, where $\theta$ is the orientation error) and satisfies world/self-collision, joint-limit, and dynamic-limit checks on a 25 ms interpolation, using the original benchmark's 0.2 rad joint-limit tolerance. We report the mean planning time (Plan), its 98th percentile (P98), the executed motion time after time-optimal parameterization (Motion), the total joint-space path length (Path, in rad), maximum absolute joint jerk per trajectory (Max Jerk, rad/s$^3$), mean absolute joint velocity over frames and joints (Mean Vel, rad/s), and maximum absolute joint acceleration per trajectory (Max Acc, rad/s$^2$). Plan and P98 use external CUDA-synchronized wall-clock solve time; all these metrics are aggregated over successful plans.

**Hand Motion Retargeting.** Following Dex-Retargeting [1], we retarget human hand motion to the Shadow robot hand and evaluate online, one frame at a time, on 501 right-hand captures from DexYCB [76]. The objective uses a keypoint position-alignment cost and a vector-alignment cost together with joint-limit constraints, and a fixed scale factor accounts for morphological differences. We report per-frame solve time, wrist-relative fingertip position error (Pos., mm), pinch and open success (%), and mean joint acceleration (Acc., rad/s$^2$). For the distilled-policy comparison against GeoRT [15] (Tab. 9) we additionally report fingertip position error (Tip), its Procrustes-aligned variant (PA-Tip), and the 95th-percentile tip error (P95), all in mm.

**Human Motion Retargeting.** We retarget full-body human motion from the SFU subset of AMASS [67] (44 sequences) to the Unitree G1 humanoid, following GMR [2]. The objective applies pose-matching costs on key body links (hands, feet, and head) together with joint-limit constraints. We report per-frame solve time, root-relative limb position error (MPJPE, cm), world-frame limb position error after one similarity alignment per 100-frame segment (WA-MPJPE, cm), as in WHAM [77], and mean joint acceleration (Acc., rad/s$^2$).

**Grasp Synthesis.** We synthesize dexterous grasps for the MANO [78] hand on objects from the test split of DexGraspNet [6], following Dex1B [7]. The objective combines the collision-avoidance cost $c_{\text{coll}}$, force-closure cost $c_{\text{fc}}$, a contact cost that encourages finger contact with the object surface, and the joint-limit cost $c_{\text{limit}}$. We synthesize $4096$ grasps ($32$ on each of $128$ test scenes) with $24$ seeds per grasp. Success requires pre-grasp penetration at most $5$ mm and passing the ManiSkill [79] grasp test under external forces, using Dex1B's force threshold; we also report the $\epsilon$-metric grasp quality (Q1) and the penetration depth (mm).

**Rendering-Based Calibration.** Following the EasyHEC [18] benchmark, we calibrate the hand-eye transform of a Panda arm by matching rendered robot masks to observed silhouettes through our coarse-to-fine rendering pipeline (Sec. 3.2). We evaluate the realistic setting with a $35$ cm / $45^\circ$ initial error and added sensor and calibration noise, and report calibration time (s), translation error (cm), and rotation error ($^\circ$).

**Physics-Based Retargeting.** Following SPIDER's [12] MuJoCo comparison, we track the human hand–object motion of 7 bimanual OakInk tasks and 3 GigaHands clips with the Ability, Allegro, Inspire, Schunk and XHand hands, five seeds each, using SPIDER's released kinematic retargets, scenes and configurations. Both methods run sampling-based MPC with the physics engine in the loop; we score $218$ of the $248$ released trials (the other $30$ fail SPIDER's own export check). We report the object's ADD-AUC (area under the curve of the object-position error over $0$–$10$ cm thresholds, as SPIDER), success, and the object's mean position (cm) and rotation ($^\circ$) errors. SPIDER's accuracy is that of its released rollouts.

**RL Training.** We train PPO [80] policies on mjlab's [20] tasks with $4096$ parallel environments: Go1 and G1 velocity tracking on flat and rough terrain, G1 motion tracking, and Yam cube lifting. RoboKit runs mjlab's rewards, terminations, commands, curricula, randomization and observations on its scenes, checked against mjlab on identical states. One evaluator scores both methods' policies: linear-velocity tracking, motion-tracking success, or time at the goal. We report wall-clock minutes to reach $95\%$ and $100\%$ of mjlab's mean final score, median over five seeds.

### A.3 Baselines

We compare RoboKit against optimization-based and learned baselines, depending on the task.

**PyRoKi** [4] is a recent kinematics optimization library built on JAX that supports batched optimization with automatic differentiation. It is our primary baseline for inverse kinematics and hand motion retargeting.

**cuRobo** [5] is a CUDA-accelerated motion-generation framework that uses L-BFGS with particle-based seeding. We compare against it on the batched IK benchmark (Tab. 7) and on trajectory optimization.

**Dex-Retargeting** [1] is a widely used library for dexterous hand retargeting, offering a position-based mode that matches joint positions and vector-based modes that align palm-to-fingertip vectors; we compare against its position and DexPilot modes. It does not support batched optimization, so we report single-instance results.

**GMR** [2] retargets human motion to humanoid robots using pose-matching objectives on keypoints subject to joint limits. It is our baseline for human motion retargeting and also runs single-instance.

**Dex1B** [7] is a large-scale grasp-synthesis benchmark that generates dexterous grasps via optimization with force-closure and contact objectives; we adopt its evaluation protocol and success criteria. We additionally compare our distilled grasping policy against the learned baselines Dex1B and UGG [73].

**GeoRT** [15] is a learned geometric hand-retargeting method, used as a learned baseline for the distilled hand-retargeting policy.

**EasyHEC** [18] is a rendering-based hand-eye calibration method and serves as our baseline for rendering-based calibration.

**SPIDER** [12] retargets human hand–object motion to dexterous hands with sampling-based MPC in MuJoCo Warp; it is our baseline for physics-based retargeting, run from its released code and configurations.

**DIAL-MPC** [11] is a sampling-based MPC with diffusion-style annealing for torque-level legged locomotion; it is our baseline for Go2 locomotion (Appendix B.3), run from its released code.

**mjlab** [20] is a GPU RL framework with Isaac Lab's manager-based API on MuJoCo Warp; it is our baseline for RL training (Tab. 4), run from its released tasks.

**Pinocchio** [81] and **Pink** [25] are widely used CPU-based Python IK libraries, which we use for the single-instance IK comparison (Tab. 12).

# B Additional Experiments

## B.1 Batch Scaling

We evaluate batch scaling for IK, motion planning, hand and humanoid retargeting, and grasp synthesis. We additionally report mean±std across 5 runs, together with IK success rates and solution errors.

Table 7: **Batched IK runtime and accuracy** on the 7-DoF Franka Panda.

| Batch | Method | Time (ms) | SR (%) | Pos. Err (mm) | Ori. Err (rad) |
|---|---|---|---|---|---|
| 1 | PyRoKi [4] | $9.08 \pm 0.72$ | $100.00 \pm 0.00$ | $(1.74 \pm 0.46) \times 10^{-4}$ | $(1.88 \pm 1.13) \times 10^{-7}$ |
| 1 | cuRobo [5] | $7.43 \pm 0.10$ | $100.00 \pm 0.00$ | $(2.13 \pm 0.94) \times 10^{-3}$ | $(8.14 \pm 4.33) \times 10^{-6}$ |
| 1 | Ours (opt) | $0.65 \pm 0.02$ | $100.00 \pm 0.00$ | $(1.23 \pm 0.45) \times 10^{-4}$ | $(1.68 \pm 1.22) \times 10^{-7}$ |
| 10 | PyRoKi [4] | $14.52 \pm 0.70$ | $100.00 \pm 0.00$ | $(3.80 \pm 7.52) \times 10^{-2}$ | $(4.86 \pm 9.66) \times 10^{-5}$ |
| 10 | cuRobo [5] | $6.73 \pm 0.02$ | $100.00 \pm 0.00$ | $(3.16 \pm 1.52) \times 10^{-3}$ | $(1.17 \pm 0.18) \times 10^{-5}$ |
| 10 | Ours (opt) | $0.61 \pm 0.02$ | $100.00 \pm 0.00$ | $(1.15 \pm 2.26) \times 10^{-2}$ | $(1.90 \pm 3.74) \times 10^{-5}$ |
| 100 | PyRoKi [4] | $16.04 \pm 0.27$ | $100.00 \pm 0.00$ | $(1.91 \pm 2.46) \times 10^{-2}$ | $(2.13 \pm 3.33) \times 10^{-5}$ |
| 100 | cuRobo [5] | $8.36 \pm 0.05$ | $99.80 \pm 0.40$ | $(4.92 \pm 1.78) \times 10^{-3}$ | $(1.39 \pm 0.09) \times 10^{-5}$ |
| 100 | Ours (opt) | $0.81 \pm 0.01$ | $100.00 \pm 0.00$ | $(1.08 \pm 0.71) \times 10^{-3}$ | $(1.20 \pm 0.82) \times 10^{-6}$ |
| 1000 | PyRoKi [4] | $48.19 \pm 1.30$ | $99.88 \pm 0.10$ | $(1.47 \pm 0.83) \times 10^{-3}$ | $(1.01 \pm 0.41) \times 10^{-6}$ |
| 1000 | cuRobo [5] | $28.48 \pm 0.70$ | $99.94 \pm 0.05$ | $(4.39 \pm 0.24) \times 10^{-3}$ | $(1.40 \pm 0.01) \times 10^{-5}$ |
| 1000 | Ours (opt) | $3.41 \pm 0.02$ | $99.94 \pm 0.05$ | $(4.04 \pm 1.53) \times 10^{-4}$ | $(4.15 \pm 0.32) \times 10^{-7}$ |
| 2000 | PyRoKi [4] | $86.20 \pm 0.73$ | $99.89 \pm 0.07$ | $(1.57 \pm 0.77) \times 10^{-3}$ | $(1.05 \pm 0.45) \times 10^{-6}$ |
| 2000 | cuRobo [5] | $50.27 \pm 0.06$ | $99.96 \pm 0.04$ | $(4.29 \pm 0.20) \times 10^{-3}$ | $(1.39 \pm 0.00) \times 10^{-5}$ |
| 2000 | Ours (opt) | $7.12 \pm 0.02$ | $99.91 \pm 0.04$ | $(3.28 \pm 0.30) \times 10^{-4}$ | $(4.03 \pm 0.25) \times 10^{-7}$ |

**Batched IK comparison with cuRobo.** Table 7 extends the main-paper IK comparison with cuRobo [5]. RoboKit is 7.1–11.5× faster than cuRobo, with comparable success rates and generally lower errors. Time covers the full batch; success requires position error below 5 mm and orientation error below 0.05 rad. Reported errors are per-run 98th percentiles over successful instances.

Table 8: **Batch latency** in milliseconds (mean±population std across 5 runs).

| Task | Method | Batch 1 | Batch 10 | Batch 100 | Batch 1000 |
|---|---|---|---|---|---|
| Motion | cuRobo | $48.61 \pm 1.06$ | $57.69 \pm 1.19$ | $335.66 \pm 1.99$ | n.m. |
| Motion | Ours (opt) | $13.27 \pm 0.57$ | $30.54 \pm 0.54$ | $230.90 \pm 0.04$ | n.m. |
| Hand | Dex-Ret. (pos) | $10.27 \pm 0.12$ | – | – | – |
| Hand | DexPilot | $17.06 \pm 0.33$ | – | – | – |
| Hand | PyRoKi | $17.72 \pm 1.99$ | $248.46 \pm 1.11$ | $648.33 \pm 3.47$ | $1106.27 \pm 132.51$ |
| Hand | Ours (opt) | $1.37 \pm 0.02$ | $1.37 \pm 0.01$ | $1.64 \pm 0.12$ | $3.02 \pm 0.09$ |
| Humanoid | GMR | $15.478 \pm 0.198$ | – | – | – |
| Humanoid | Ours (opt) | $2.212 \pm 0.002$ | $2.293 \pm 0.003$ | $3.034 \pm 0.050$ | $13.129 \pm 0.123$ |
| Grasp | Ours (LM-mesh) | $201.67 \pm 20.02$ | $275.31 \pm 8.70$ | $592.86 \pm 3.62$ | $4088.45 \pm 2.52$ |

**Batch latency across other tasks.** Table 8 shows efficient scaling for online retargeting: increasing the batch from 1 to 1000 raises RoboKit's hand latency from 1.37 to 3.02 ms and humanoid latency from 2.21 to 13.13 ms. RoboKit is faster than PyRoKi at every measured hand batch size, while Dex-Retargeting and GMR lack batched APIs. With four trajectory candidates per query for both methods, RoboKit is faster than cuRobo at every measured motion-planning batch size. Grasp optimization processes 1000 initializations in 4.09 s.

Online timings are per frame across all streams; motion and grasp timings cover the full batch. These latency sweeps use different workloads from Tab. 2. cuRobo's graph search is disabled to retain four candidates. A dash denotes an unavailable batched API; n.m. denotes an unmeasured batch size.

### B.2 Comparison with Learned Policies

Beyond the optimization baselines in the main paper, we compare our distilled policies against dedicated learned methods. For hand motion retargeting, we evaluate against GeoRT [15] on DexYCB data.

Table 9: **Hand retargeting** against GeoRT [15] on DexYCB.

| Method | Tip (mm) ↓ | PA-Tip (mm) ↓ | P95 (mm) ↓ |
|---|---|---|---|
| GeoRT [15] | 37.4 | 9.5 | 58.5 |
| Ours (distilled) | **23.0** | **7.8** | **29.9** |

**Comparison with learned policies.** We report fingertip position error (Tip), Procrustes-aligned tip error (PA-Tip), and the 95th-percentile tip error (P95), all in mm.

Our distilled policy, trained with geometry-aware retargeting costs, outperforms GeoRT on all three metrics (Tab. 9), reducing the fingertip position error from 37.4 to 23.0 mm and the 95th-percentile error from 58.5 to 29.9 mm. For grasp synthesis, our distilled policy already surpasses the learned baselines UGG [73] and Dex1B [7] (see the main paper).

### B.3 Comparison with DIAL-MPC

We compare against DIAL-MPC [11], a sampling-based MPC for torque-level legged locomotion, on the three Go2 tasks of its Table I: velocity tracking with a trot gait, a sequence of 10 random jumps, and climbing onto a crate 0.125–0.6 m tall (10 heights, two seeds each). Both methods use DIAL-MPC's released models, rewards and control settings. RoboKit uses one configuration for all three tasks, with $0.6\times$ DIAL-MPC's samples. We report the tracking error, DIAL-MPC's jump score, and crate success (all four feet on top, torso upright). Time is wall-clock per control step.

Table 10: **Locomotion MPC** against DIAL-MPC [11] on Go2; time per control step.

| Method | Trot | | Seq. Jump | | Crate Climb | |
|---|---|---|---|---|---|---|
| | Track. err. ↓ | Time (ms) ↓ | Score ↑ | Time (ms) ↓ | Succ. (%) ↑ | Time (ms) ↓ |
| DIAL-MPC [11] | 0.0030 | 74.4 | 0.885 | 65.7 | 75 | 1354 |
| Ours | **0.0025** | **34.8** | **0.920** | **29.1** | **80** | **163** |

RoboKit matches or exceeds DIAL-MPC on all three tasks while running $2.1\times$, $2.3\times$ and $8.3\times$ faster (Tab. 10).

### B.4 Grasp Synthesis Ablation

We compare mesh and SDF geometry representations for grasp synthesis on $4{,}096$ grasps ($128$ scenes $\times$ $32$ initializations).

Table 11: **Grasp synthesis ablation** on $4{,}096$ grasps (mean$\pm$population std across $5$ runs).

| Method | Time (s) | SR (%) | Q1 | Pen. (mm) |
|---|---|---|---|---|
| LM-mesh | $16.605 \pm 0.029$ | $48.49 \pm 0.56$ | $0.0702 \pm 0.0004$ | $0.547 \pm 0.007$ |
| GD-mesh | $18.412 \pm 0.378$ | $41.89 \pm 0.89$ | $0.0683 \pm 0.0013$ | $1.441 \pm 0.017$ |
| LM-sdf | $2.973 \pm 0.033$ | $46.83 \pm 0.41$ | $0.0666 \pm 0.0007$ | $0.702 \pm 0.014$ |
| GD-sdf | $4.334 \pm 0.007$ | $41.52 \pm 0.78$ | $0.0657 \pm 0.0012$ | $1.485 \pm 0.017$ |

**Grasp synthesis ablation.** Values are mean$\pm$population std across $5$ runs, using the original full evaluation. Time covers optimizer iterations; success requires pre-grasp penetration at most $5$ mm and passing the ManiSkill grasp test under external forces. LM uses $64$ iterations and annealed GD uses $6{,}000$.

### B.5 Comparison to CPU IK Baselines

We compare our solver against Pinocchio [81] and Pink [25], two widely-used CPU-based Python IK libraries, using their official example scripts (Tab. 12).

Table 12: **Comparison to CPU IK baselines** on the 7-DoF Franka Panda arm.

| Method | Time (ms) | SR (%) | Pos. Err (mm) | Ori. Err (rad) |
|---|---|---|---|---|
| Pinocchio [81] | 3.35 | 95.31 | 1.923 | $4.53 \times 10^{-3}$ |
| Pink [25] | 10.80 | 79.69 | $3.95 \times 10^{-4}$ | $4.28 \times 10^{-7}$ |
| Ours (opt) | **2.36** | **100** | $4.05 \times 10^{-4}$ | $4.78 \times 10^{-7}$ |

**Comparison to CPU IK baselines.** Our solver leverages the GPU for random seed sampling and the Levenberg–Marquardt iterations. Both our method and Pink enforce joint-limit constraints, whereas the default Pinocchio IK example does not. Thanks to multi-seed sampling, our method achieves a $100\%$ success rate, compared to $95.31\%$ for Pinocchio and $79.69\%$ for Pink, while also being the fastest at $2.36$ ms ($1.4\times$ faster than Pinocchio, $4.6\times$ faster than Pink). Solution accuracy is comparable to Pink and roughly an order of magnitude better than Pinocchio in both position and orientation error.

# C Implementation Details

## C.1 Analytic Objectives

Table 13 summarizes the pre-built analytic objectives; we detail their mathematical formulations below.

Table 13: **Pre-built analytic objectives.**

| Term | Purpose | Analytic Jacobian |
|---|---|---|
| *Task Objectives* | | |
| Pose Matching | Match link SE(3) pose to target via log map | ✓ |
| Retargeting | Pairwise link-to-keypoint alignment | ✓ |
| Contact Attraction | Attract collision spheres to object contact points | ✓ |
| Force Closure | Penalize net contact wrench for grasp equilibrium | ✓ |
| Mask Alignment | Match rendered robot masks to observed silhouettes | ✓* |
| *Safety & Collision* | | |
| Collision Avoidance | Penalize robot sphere penetration into mesh SDF | ✓ |
| Self-Collision | Penalize inter-link sphere–sphere penetration | ✓ |
| *Joint & Derivative Limits* | | |
| Joint Limit | Penalize joint positions outside mechanical bounds | ✓ |
| Velocity Limit | Penalize joint velocities exceeding limits | ✓ |
| Acceleration/Jerk Limits | Penalize trajectory joint derivatives outside bounds | ✓ |
| *Regularization & Smoothness* | | |
| Smoothness | Penalize joint/base changes between consecutive steps | ✓ |
| EE Pose Smoothness | Penalize EE SE(3) pose changes across trajectory | ✓ |
| Rest Pose | Regularize joints toward a rest configuration | ✓ |
| Start Config | Penalize deviation from initial trajectory config | ✓ |
| Base Damping | Penalize floating-base motion from reference pose | ✓ |
| *Performance* | | |
| Manipulability | Maximize Yoshikawa manipulability at target link | ✓ |

All terms support batched evaluation. Mask alignment ($*$) combines analytical projection derivatives with central-difference image gradients; the other listed terms use analytical Jacobians.

**Mathematical Formulations** The solver minimizes the sum of squared weighted residuals:

$$\min_q \ \frac{1}{2} \sum_k \left\| W_k \, c_k(q) \right\|_2^2,$$

where $c_k(q)$ is an unweighted residual vector and $W_k = \mathrm{diag}(\alpha_k \, w_k)$ combines the task gain $\alpha_k \in (0, 1]$ with per-dimension residual weights $w_k$. Task inputs are omitted here for brevity.

**Pose Matching Cost.** The pose matching cost penalizes the SE(3) error between the current link pose and a target:

$$c_{\mathrm{pose}}(q, \, i, \, {}^{w}\mathbf{T}_{\mathrm{tgt}}) = \log\!\left( {}^{w}\mathbf{T}_i^{-1} \, {}^{w}\mathbf{T}_{\mathrm{tgt}} \right) \in \mathbb{R}^6,$$

where $\log : \mathrm{SE}(3) \to \mathfrak{se}(3)$ is the logarithm map. The first three components are weighted by $w_{\mathrm{pos}}$ and the last three by $w_{\mathrm{ori}}$. The analytical Jacobian is:

$$\frac{\partial c_{\mathrm{pose}}}{\partial q} = - \, J_{\log}\!\left( {}^{w}\mathbf{T}_{\mathrm{tgt}}^{-1} \, {}^{w}\mathbf{T}_i \right) \mathrm{Ad}({}^{w}\mathbf{T}_i^{-1}) \, \mathbf{S}_i,$$

where $J_{\log}$ is the Jacobian of the SE(3) log map, $\mathrm{Ad}(\cdot)$ is the adjoint representation, and $\mathbf{S}_i$ is the world-frame geometric Jacobian of link $i$.

**Retargeting Cost.** The retargeting cost measures pairwise correspondence between robot link positions $\{p_k^{\mathrm{r}}\}$ and target keypoints $\{p_k^{\mathrm{t}}\}$. For each pair $(i, j)$:

$$c_{\mathrm{retarget}}(q) = \left( p_i^{\mathrm{t}} - p_j^{\mathrm{t}} \right) - s \cdot \left( p_i^{\mathrm{r}} - p_j^{\mathrm{r}} \right),$$

where $s$ is a position scale factor. An optional angular term penalizes direction mismatch of the displacement vectors.

**Contact Attraction Cost.** The contact attraction cost encourages robot collision spheres to reach object contact points:

$$c_{\text{contact}}(q,\, p_{\text{obj}}) = \max\!\big(0,\ \min_i\big(\|p_{\text{obj}} - {}^{w}c_i\| - r_i\big)\big),$$

where ${}^{w}c_i$ and $r_i$ are the world-frame center and radius of the $i$-th collision sphere. The Jacobian propagates through only the closest active sphere via $-\hat{d}^{\top}\frac{\partial {}^{w}c_{i^*}}{\partial q}$, where $\hat{d}$ is the unit direction from sphere to contact point.

**Force Closure Cost.** The force closure cost drives the net contact wrench toward zero, encouraging the grasp to resist arbitrary external loads:

$$c_{\text{fc}}(q) = \begin{bmatrix} \sum_k \hat{n}_k \\ \sum_k p_k \times \hat{n}_k \end{bmatrix} \in \mathbb{R}^6,$$

where $\hat{n}_k$ is the object-surface unit normal at the $k$-th contact (queried dynamically from the object geometry) and $p_k$ is the world-frame contact position. The first three components penalize net force imbalance and the last three penalize net torque imbalance. With $N_k = \partial \hat{n}_k / \partial p_k$, the analytical Jacobian is:

$$\frac{\partial c_{\text{fc}}}{\partial q} = \begin{bmatrix} \sum_k N_k \frac{\partial p_k}{\partial q} \\ \sum_k \left( \frac{\partial p_k}{\partial q} \times \hat{n}_k + p_k \times N_k \frac{\partial p_k}{\partial q} \right) \end{bmatrix}.$$

Voxel queries differentiate the normalized trilinear SDF gradient analytically, and primitive queries use analytical normal derivatives. Mesh queries return triangle face normals, so $N_k = 0$ within the selected face. Derivatives are discontinuous at voxel-cell, face, and closest-object switches; no finite-difference normal queries are used.

**Mask Alignment Cost.** For rendered and observed masks $M^{\text{r}}$ and $M^{\text{o}}$ at the current pyramid level, including its configured blur, the per-pixel residual is:

$$c_{\text{mask},uv} = M^{\text{r}}_{uv} - M^{\text{o}}_{uv}.$$

For a camera-frame twist $\boldsymbol{\xi}$, the image-motion Jacobian is:

$$\frac{\partial c_{\text{mask},uv}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} \partial_u M^{\text{r}} & \partial_v M^{\text{r}} \end{bmatrix} \frac{\partial (u, v)}{\partial \boldsymbol{\xi}}.$$

The projection derivative is analytical; image gradients use central pixel differences, e.g., $\partial_u M^{\text{r}}_{uv} = (M^{\text{r}}_{u+1,v} - M^{\text{r}}_{u-1,v})/2$ for interior pixels. An SE(3) adjoint maps the camera-frame Jacobian to the calibration variable's tangent frame. Per-observation weights scale the squared residuals, and the implementation accumulates $J^{\top}J$ and $J^{\top}c$ over pixels without storing the full Jacobian. Optional valid depth observations add a Huber residual.

**Collision Avoidance Cost.** The robot is approximated by collision spheres $s_i = (c_i, r_i)$, and each object $\mathcal{O}$ by the hybrid voxel-SDF and mesh representation (Sec. 3.2), which gates an exact mesh query to a near-surface band and otherwise reads a cheap baked voxel SDF:

$$c_{\text{coll}}(q) = \max\!\big(0,\ r_i + \epsilon - \text{SDF}({}^{w}c_i,\, \mathcal{O})\big),$$

where $\text{SDF}(\cdot, \mathcal{O})$ returns the signed distance to object $\mathcal{O}$ (negative inside), and $\epsilon \geq 0$ is a safety margin. When multiple objects are present, $\text{SDF}_{\text{eff}} = \min_{\mathcal{O}} \text{SDF}(\cdot, \mathcal{O})$. The Jacobian is $-\hat{n}_i^{\top}\frac{\partial {}^{w}c_i}{\partial q}$ when the constraint is active, where $\hat{n}_i$ is the outward surface normal and $\frac{\partial {}^{w}c_i}{\partial q} = \mathbf{v} + \boldsymbol{\omega} \times {}^{w}c_i$ from the spatial motion subspace.

**Self-Collision Cost.** For non-adjacent link pairs $(i, j)$, the self-collision cost penalizes sphere overlap:

$$c_{\text{self}}(q) = \max\!\big(0,\ r_i + r_j + \epsilon - \|{}^{w}c_i - {}^{w}c_j\|\big).$$

The Jacobian uses the unit direction between sphere centers: $-\hat{d}_{ij}^{\top}(\frac{\partial {}^{w}c_i}{\partial q} - \frac{\partial {}^{w}c_j}{\partial q})$.

**Joint Limit Cost.** The joint limit cost penalizes configurations exceeding mechanical bounds:

$$c_{\text{limit}}(q) = \max(0,\, q - q_{\text{upper}}) + \max(0,\, q_{\text{lower}} - q).$$

The Jacobian is a diagonal matrix of signs: $+1$ above the upper limit, $-1$ below the lower limit, and $0$ otherwise. Hard QP constraints are also available: $q_{\text{lower}} - q \le \Delta q \le q_{\text{upper}} - q$.

**Velocity Limit Cost.** The velocity limit cost penalizes excessive joint velocities:

$$c_{\text{vel}}(q_t, q_{t-1}) = \max\left(0,\, \left|\frac{q_t - q_{t-1}}{\Delta t}\right| - \dot{q}_{\max}\right).$$

The Jacobian is $\frac{\text{sign}(\dot{q})}{\Delta t}$ when the limit is violated.

**Acceleration and Jerk Limit Costs.** For derivative order $p \in \{2, 3\}$, trajectory limits penalize acceleration or jerk outside per-joint bounds $[l_j^{(p)}, u_j^{(p)}]$:

$$d_{t,j}^{(p)} = \frac{1}{\Delta t^p} \sum_{k=-2}^{2} \beta_k^{(p)} q_{t+k,j}, \qquad c_{t,j}^{(p)} = \max\Big(0,\, l_j^{(p)} - d_{t,j}^{(p)},\, d_{t,j}^{(p)} - u_j^{(p)}\Big).$$

The default five-point central stencils, ordered from $k = -2$ to 2, are:

$$\boldsymbol{\beta}^{(2)} = \tfrac{1}{12}(-1, 16, -30, 16, -1), \qquad \boldsymbol{\beta}^{(3)} = (-\tfrac{1}{2}, 1, 0, -1, \tfrac{1}{2}).$$

Boundary indices are clamped to the first or last frame, holding endpoint values. Each sparse Jacobian entry is $\sigma_{t,j}^{(p)} \beta_k^{(p)} / \Delta t^p$, where $\sigma = -1$ below the lower bound, $+1$ above the upper bound, and $0$ otherwise; entries mapping to the same held frame add together. Forward-difference stencils are also supported. Revolute-joint acceleration and jerk bounds have units rad/s$^2$ and rad/s$^3$, respectively.

**Smoothness Cost.** The smoothness cost penalizes large joint changes between consecutive time steps:

$$c_{\text{smooth}}(q_t, q_{t-1}) = q_t - q_{t-1}.$$

For floating-base robots, an additional base term is $\log({}^w\mathbf{T}_{b,t}\, {}^w\mathbf{T}_{b,t-1}^{-1})$. The joint Jacobian is the identity matrix; the base Jacobian is $J_{\log}(T_{\text{error}}) \cdot \text{Ad}({}^w\mathbf{T}_{b,t})$.

**EE Pose Smoothness Cost.** Penalizes changes in end-effector SE(3) poses between consecutive trajectory frames:

$$c_{\text{ee}}(t, k) = \log\big({}^w\mathbf{T}_{k,t}^{-1}\, {}^w\mathbf{T}_{k,t-1}\big) \in \mathbb{R}^6,$$

for each end-effector $k$ and consecutive frames $t$, $t-1$.

**Rest Pose Cost.** The rest pose cost regularizes joints toward a preferred configuration:

$$c_{\text{rest}}(q) = q - q_{\text{rest}}.$$

For floating-base robots, an additional term is $\log({}^w\mathbf{T}_b\, {}^w\mathbf{T}_{\text{rest}}^{-1})$. The joint Jacobian is the identity matrix.

**Start Configuration Cost.** For trajectory optimization, penalizes deviation of the first frame from a specified initial configuration: $c_{\text{start}}(q_0) = q_0 - q_{\text{init}}$.

**Base Damping Cost.** Discourages floating-base motion from a reference pose:

$$c_{\text{damp}}({}^w\mathbf{T}_b) = \log\big({}^w\mathbf{T}_b\, {}^w\mathbf{T}_{\text{ref}}^{-1}\big) \in \mathbb{R}^6.$$

**Manipulability Cost.** Maximizes the Yoshikawa measure at a target link:

$$m(q) = \sqrt{\det\big(\mathbf{J}_p\, \mathbf{J}_p^\top\big)}, \qquad c_{\text{manip}}(q) = \frac{1}{m(q) + \varepsilon},$$

where $\mathbf{J}_p \in \mathbb{R}^{3\times n}$ is the translational Jacobian at the target link. The analytical gradient uses the cofactor matrix of $\mathbf{J}_p\mathbf{J}_p^\top$ and the derivative of $\mathbf{J}_p$ through the kinematic chain via spatial adjoints.

### C.2 Algorithm Summary

We summarize the full RoboKit pipeline in three stages: parallel kinematic optimization (Algorithm 1), offline policy distillation from filtered demonstrations (Algorithm 2), and online policy fine-tuning through differentiable objectives (Algorithm 3). Algorithm 4 combines these stages sequentially: offline supervised training on filtered demonstrations initializes the policy, followed by fine-tuning with the online objective on freshly sampled task instances.

**Algorithm 1** Parallel Geometry-Aware Kinematic Solving

**Require:** Residuals and weights $\{(c_k, W_k)\}$, task distribution $p(o)$, number of particles $M$, LM iterations $T_{\text{opt}}$, number of rounds $R$, top-$K$ selection ratio
**Ensure:** Solution dataset $\hat{\mathcal{D}} = \{(o_i, q_i^*)\}$
**for** each task instance $o_i \sim p(o)$ **do**
  Sample $M$ initial particles $\{q^{(m)}\}_{m=1}^{M}$
  **for** $r = 1$ **to** $R$ **do**
    **for** $t = 1$ **to** $T_{\text{opt}}$ **do**
      Evaluate weighted residuals $r_k^{(m)} = W_k\, c_k(q^{(m)}; o_i)$ for all $k, m$
      Compute analytical sparse Jacobians $J_k^{(m)}$
      Update each particle via Levenberg–Marquardt step
    **end for**
    Rank particles by cost: $\mathcal{S}^{(m)} = \frac{1}{2}\sum_k \|r_k^{(m)}\|_2^2$
    Select top-$K$ particles $\{q^{(m)}\}_{m=1}^{K}$
    **if** $r < R$ **and** resampling enabled **then**
      Resample $M$ particles from top-$K$
    **end if**
  **end for**
  Select best solution: $q_i^* = \arg\min_m \mathcal{S}^{(m)}$
  Store $(o_i, q_i^*)$ in $\hat{\mathcal{D}}$
**end for**
**return** $\hat{\mathcal{D}}$

**Algorithm 2** Offline Policy Distillation

**Require:** Candidate dataset $\hat{\mathcal{D}}$ from Algorithm 1, filtering criterion $\mathcal{F}$, policy $\pi_\theta$, learning rate $\eta$, training iterations $T_{\text{off}}$
**Ensure:** Policy $\pi_\theta$
Filter: $\mathcal{D} = \{(o_i, q_i^*) \in \hat{\mathcal{D}} \mid \mathcal{F}(o_i, q_i^*) = \text{True}\}$
**for** $t = 1$ **to** $T_{\text{off}}$ **do**
  Sample mini-batch $(o, q^*) \sim \mathcal{D}$
  Compute $\mathcal{L}_{\text{offline}}(\theta) = \|\pi_\theta(o) - q^*\|^2$
  Update $\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}_{\text{offline}}(\theta)$
**end for**
**return** $\pi_\theta$

**Algorithm 3** Online Policy Fine-Tuning

**Require:** Residuals and weights $\{(c_k, W_k)\}$, task distribution $p(o)$, pretrained policy $\pi_\theta$ from Algorithm 2, learning rate $\eta$, training iterations $T_{\text{on}}$
**Ensure:** Policy $\pi_\theta$
**for** $t = 1$ **to** $T_{\text{on}}$ **do**
  Sample task instances $o \sim p(o)$
  Predict $\hat{q} = \pi_\theta(o)$
  Compute objective loss $\mathcal{L}(\theta) = \frac{1}{2}\sum_k \|W_k\, c_k(\hat{q}; o)\|_2^2$
  Update $\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}(\theta)$
**end for**
**return** $\pi_\theta$

**Algorithm 4** Hybrid Policy Distillation (Offline Warm-Up, Online Fine-Tuning)

**Require:** Candidate dataset $\hat{\mathcal{D}}$ from Algorithm 1, filtering criterion $\mathcal{F}$, residuals and weights $\{(c_k, W_k)\}$, task distribution $p(o)$, policy $\pi_\theta$, learning rate $\eta$, offline iterations $T_{\text{off}}$, online iterations $T_{\text{on}}$

**Ensure:** Policy $\pi_\theta$

1: Train $\pi_\theta$ on filtered demonstrations using Algorithm 2 for $T_{\text{off}}$ iterations
2: Fine-tune the resulting $\pi_\theta$ with the online objective using Algorithm 3 for $T_{\text{on}}$ iterations
3: **return** $\pi_\theta$

## D Additional Visualization

We provide additional visualization in Fig. 4.




Figure 4: Qualitative results of RoboKit across core tasks. A single composite figure showcasing hand motion retargeting, grasp synthesis, trajectory optimization, and human motion retargeting.