Batched Block-Banded GPU Solvers for Whole-Trajectory Inverse Kinematics

Dylan Turpin[1]

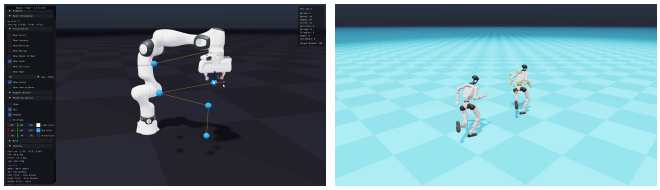

Fig. 1. **Left:** a 90-frame pick-and-place trajectory re-optimized jointly at interactive rates while a waypoint is dragged (Franka FR3). **Right:** a motion-capture dance clip retargeted to a floating-base 29-DoF Unitree G1 by solving all frames of the clip in one shot (left robot; markers show effector targets), next to the frame-by-frame production baseline (right robot).

***Abstract*— Kinematic pipelines that produce robot joint trajectories from task-space targets—motion retargeting, demonstration synthesis, interactive motion authoring—typically solve inverse kinematics (IK) frame by frame: a greedy, sequential process that cannot exploit GPU parallelism across time and that produces trajectories with large frame-to-frame accelerations. We extend a batched GPU IK solver to optimize *all frames of one or more trajectories jointly* as a single nonlinear least-squares problem. Temporal objectives (finite-difference velocity/acceleration/jerk smoothness, velocity limits, reference attraction) report bounded per-frame-offset Jacobian blocks, so the Gauss–Newton system is assembled directly in block-banded form without ever materializing a global Jacobian. We implement and compare three GPU linear backends for the banded normal equations, all expressed with GPU tile primitives and all CUDA-graph capturable: (i) a batched block-tridiagonal Cholesky (one thread block per trajectory), (ii) a SPIKE-style parallel-in-time direct solver that factorizes partitions of the horizon concurrently and couples them through a symmetric positive-definite Schur system on interface blocks, and (iii) a block-Jacobi preconditioned conjugate-gradient on a block-sparse matrix. On a Franka FR3 tracking task (120 frames, four trajectories) the joint solve is $40\times$ faster than warm-started sequential per-frame IK at equal iteration budgets while producing $2.9\times$ lower peak acceleration, and $12\times$ faster than an equivalent jaxls/PyRoKi formulation of the same problem that reaches the same solution. The parallel-in-time backend keeps the exactness of the sequential direct sweep while cutting its long-horizon latency by $5.8\times$ at $T{=}960$. A humanoid case study retargets motion-capture clips to a floating-base 29-DoF Unitree G1, replacing a production frame-by-frame retargeting loop with a joint solve that is two orders of magnitude faster and removes its acceleration spikes, and retargets the 288-hour BONES-SEED corpus in 2.7 h on one GPU. At the batch-one extreme, a million-frame engraving toolpath — 2.4 h of execution — solves as a single trajectory in 1.1 s.**

I. Introduction

Inverse kinematics on the GPU has recently become fast enough to treat as a throughput primitive: modern batched solvers answer thousands of single-frame queries in milliseconds [1], [2]. Yet most consumers of IK do not want a single frame—they want a *trajectory*. Motion retargeting maps a human motion clip onto a robot skeleton frame by frame [3]; demonstration-synthesis pipelines roll task scripts into joint trajectories for policy training [4]–[6]; interactive authoring tools re-solve a whole motion while a user drags a waypoint. The dominant implementation strategy inherits the single-frame primitive: warm start from the previous frame and solve frames sequentially. This is doubly wasteful on a GPU. First, it serializes over time, so a 1000-frame clip pays a thousand launch-and-converge round trips regardless of how parallel each solve is. Second, it is *greedy*: each frame minimizes its own error with no knowledge of the future, so the concatenated solution exhibits large accelerations and jerk that must be cleaned up by post-processing.

The alternative is classical: pose the whole trajectory as one nonlinear least-squares problem with per-frame task costs and temporal coupling costs. As Toussaint's k-order Markov formulation (KOMO) makes explicit, the Gauss–Newton Hessian of such a problem is *block-banded*— block-tridiagonal for first-difference costs, wider for acceleration and jerk—and the Newton step costs $O(T\,k^2 n^3)$, linear in the horizon T [7]. This is the same structure exploited by Riccati sweeps in DDP [8], [9] and by factor-graph elimination in GPMP2 [10], [11]. What is missing is a GPU treatment of the *banded direct solve*: existing GPU trajectory optimizers either avoid the linear system altogether via quasi-Newton methods on low-dimensional spline knots [1], [12], or solve the normal equations iteratively [2], [13], paying the conditioning tax of conjugate gradients.

We present a trajectory extension of the IK solver in the Newton physics engine, built on Warp's tile primitives [14], that solves all frames of one or more joint trajectories jointly and treats the banded linear solve as a first-class design axis. Our contributions:

- **Trajectory IK with band-direct assembly.** Existing per-frame objectives (end-effector pose, joint limits) are reused unchanged with one target per frame; new temporal objectives report bounded per-frame-offset Jacobian coefficient blocks that are accumulated analytically into a banded store—no global Jacobian is ever materialized. Root free-joint tangents are body-centered (rotations pivot about the base's own anchor), which we found necessary for floating-base convergence away from the world origin.
- **Three banded GPU backends, one interface.** A batched block-tridiagonal Cholesky (block Thomas; one

[1]NVIDIA. dturpin@nvidia.com

CUDA thread block per trajectory, sequential over frames inside a single tile kernel); a SPIKE-style [15] parallel-in-time direct solver whose symmetric Schur-complement reduction preserves positive definiteness so every factorization remains a Cholesky; and a block-Jacobi preconditioned conjugate-gradient on a block-sparse (BSR) matrix. Levenberg–Marquardt globalization [16], [17] is per-trajectory, fully device-resident, and the entire optimizer step is CUDA-graph capturable for all three backends.

- **An explicit timing taxonomy and cross-library comparison.** We separate one-time compilation, cold solves, and steady-state (graph-replay) solves, map competing systems' published protocols onto the same taxonomy, and compare against sequential warm-started IK, an equivalent jaxls/PyRoKi formulation of the identical problem, and cuRobo's trajectory optimizer as context, plus a humanoid motion-retargeting case study on a 29-DoF Unitree G1.

II. Related Work

GPU inverse kinematics. cuRobo [1] popularized batched GPU IK: thousands of seeds optimized in parallel with a fused L-BFGS and CUDA-graph capture. Its successor cuRoboV2 [12] extends to whole-body multi-link IK for high-DoF humanoids and ships a *MotionRetargeter* that is explicitly frame-by-frame warm-started IK or MPC—the sequential-in-time pattern this paper replaces—and is productized as cuMotion [18]. PyRoKi [2] offers a modular JAX kinematic-optimization toolkit with analytic-Jacobian costs and Levenberg–Marquardt on factor graphs [19], [20]. Neither system provides a banded *direct* solver for trajectory-level problems: jaxls defaults to conjugate gradients (its only sparse direct path is a CPU host-callback), and cuRobo never forms a Jacobian, embedding temporal structure in a low-dimensional B-spline instead.

Trajectory optimization structure. That trajectory costs with bounded temporal stencils yield banded systems is long established: KOMO solves the banded system with a sequential CPU Cholesky [7]; CHOMP's smoothness metric is a banded operator [21]; STOMP [22] and predictive sampling [23] sidestep the linear algebra by sampling; TrajOpt uses general sparse SQP [24]; GPMP2 eliminates a factor graph with the same sparsity [10], [11]; DDP-family solvers factorize it as a Riccati recursion [8], [9]. Recent GPU trajectory optimizers include MPCGPU, which solves the block-tridiagonal Schur KKT system with preconditioned conjugate gradients [13]; GATO, which batches whole DDP-style solves across problems [25]; CusADi, which code-generates batched optimal-control evaluations [26]; and parallel-in-time sequential convex programming [27]. Vectorized sampling planners make the batching argument from the sampling side [28]. Our work differs in combining *exact* banded direct solves with both batch parallelism (over trajectories) and parallelism in time (within a trajectory) inside one LM trajectory-IK solver.

Parallel-in-time linear algebra. Direct parallel solvers for (block-)tridiagonal systems date to block cyclic reduction [29]; SPIKE partitions a banded matrix into independent diagonal blocks coupled by interface unknowns [15], [30]. Associative-scan formulations achieve $O(\log T)$ span for Bayesian smoothing and LQR [31], [32]. We adopt a SPIKE-style partitioning but form the reduced system as a Schur complement on single-block separators, which—unlike the classic SPIKE reduced system—stays symmetric positive definite, so the whole pipeline remains Cholesky-based (Sec. IV-B).

Humanoid retargeting. Kinematic retargeting is the upstream stage of current humanoid teleoperation and tracking stacks [4]–[6], and retargeting quality measurably dominates downstream tracking performance [3]. Production retargeting today is frame-by-frame IK (e.g., cuRoboV2's MotionRetargeter and GMR [3]); our case study solves whole clips jointly.

III. Trajectory IK Formulation

A. Problem

Let $q_t \in \mathbb{R}^{n_q}$ be the joint configuration at frame $t \in \{1, \dots, T\}$ of a trajectory, with n tangent-space degrees of freedom, and let B trajectories be optimized simultaneously. We minimize

$$\min_{q_{1:T}^{(1:B)}} \sum_{b,t} \tfrac{1}{2} \left\| r_{\text{task}}\left(q_t^{(b)};\, g_t^{(b)}\right) \right\|^2 + \sum_{b,t} \tfrac{1}{2} \left\| r_{\text{temp}}\left(q_{t:t+k}^{(b)}\right) \right\|^2, \quad (1)$$

where per-frame residuals r_{task} comprise end-effector position and orientation errors against per-frame targets g_t and joint-limit penalties, and temporal residuals r_{temp} couple at most $k{+}1$ consecutive frames. Updates are computed in velocity (tangent) space and applied through the same retraction as the single-frame solver, which handles quaternion (ball and free) joints uniformly.

B. Temporal objectives with bounded stencils

A temporal objective contributes n residual rows per frame. Smoothness of finite-difference order $d \in \{1, 2, 3\}$ (velocity, acceleration, jerk) uses the alternating binomial stencil scaled by $w/\Delta t^d$; quaternion joints use tangent-space (log-map) differences consistent with the solver's retraction, and per-DoF weights allow, e.g., excluding the floating base of a retargeted humanoid whose root motion is signal rather than noise. Velocity-limit and reference-attraction objectives follow the same contract. Rather than emitting rows of a global Jacobian, each temporal objective reports coefficient blocks $\partial r_t / \partial u_{t+j} \in \mathbb{R}^{n \times n}$ for offsets $j \in \{0, \dots, k\}$. Most joints populate only the diagonal of each block. Root free-joint tangents are body-centered — the retraction translates the base anchor by δ_{lin} and rotates about it — so their rows are diagonal too and convergence is invariant to where a trajectory sits in the world (a world-origin pivot makes the effector Jacobian error grow with distance and stalls the solve meters out). Non-root free-joint linear rows carry the exact $[p]_\times$ lever-arm coupling of their parent-anchor retraction; all blocks are finite-difference verified, including at multi-meter world offsets.

C. Dynamics-aware temporal objectives

The same contract extends from kinematic to dynamic costs; the band store accepts fully dense $n \times n$ blocks, so no assembly changes are needed.

Gravity torque. A stencil-0 objective penalizes the static holding torque $\tau_g(q) = \partial U / \partial q$ (the gravity bias of the manipulator equation), steering redundant DoFs toward gravity-friendly postures. Writing $u_d = M_d\, v_d + \omega_d \times P_d$, where (v_d, ω_d) is DoF d's world motion subspace and M_d, P_d are the total mass and mass-weighted CoM of the bodies it moves, the residual is $\tau_{g,d} = -g \cdot u_d$ and the dense block is the analytic gravity Hessian: for an ancestor DoF a and descendant d on one chain, $\partial \tau_{g,d} / \partial u_a = -g \cdot (\omega_a \times u_d)$, symmetric in (a, d) and zero across branches — exact for revolute, prismatic, and free-joint tangents. Floating-base DoFs are unactuated: their residual rows are zero while their columns (how base motion changes actuated torques) are kept. Residuals match Newton's `eval_inverse_dynamics` gravity force to 10^{-7}, and blocks are finite-difference verified.

Apparent gravity (the waiter problem). A carried surface keeps an object resting on it when the apparent specific force $f_t = a_{\text{ee},t} - g$ stays aligned with the surface normal. A stencil-2 objective takes a_{ee} as the forward second difference of the carried point over frames $t..t{+}2$ and penalizes the two surface-tangential components of f_t, with tangent axes evaluated mid-stencil. Its blocks combine the acceleration stencil projected on the tangent axes with a surface-tilt term $(\omega_a \times t_k) \cdot f_t$, so the solver both smooths the carried point's acceleration *and banks the surface into the motion*. This is the task-space temporal cost anticipated in our design notes: the two residual rows are simply padded into the n-row temporal block, and the existing dense-block assembly carries the $J_{\text{task}}^\top J_{\text{task}}$ coupling without a contract extension.

Retargeting objectives. The same patterns port PyRoKi's retargeting costs [2]: smoothed one-sided ground-plane penalties on selected link points and on link-attached capsule surfaces (the collision smoothing of [1]), contact-gated stance pins and a frame-to-frame foot-skate lock (contact labeled on the source motion), and capsule-pair self-collision separation. Fused multi-effector position/rotation objectives evaluate all effector rows in two kernels, addressing the per-iteration assembly cost that dominates at many effectors. We deliberately do not port PyRoKi's per-frame pairwise-scale parameterization: it introduces non-joint decision variables, and the production pipeline we compare against maps a uniform-proportion skeleton through a fixed scaler.

D. Band assembly and Levenberg–Marquardt

The Gauss–Newton normal equations of (1) are block-banded with bandwidth $2(k{+}1)n - 1$ (Fig. 2). Per-frame objectives contribute block-diagonal terms $J_t^\top J_t$ computed with dense tile kernels reused from the single-frame solver (analytic, autodiff, or mixed Jacobians); temporal objectives accumulate $H(t, t{+}j)$ $+\!=$ $\sum_i C_i^\top C_{i+j}$ analytically into a banded store of $k{+}1$ blocks per frame. Damping λ and step acceptance are *per trajectory*: one trust-region ratio decides atomically whether all T frames of a trajectory accept the proposed update, following the single-frame solver's LM schedule. All state is device-resident; a fixed iteration count makes the entire optimizer step CUDA-graph capturable for all backends. Frames may be pinned exactly (e.g., anchoring frame 0 at the current robot state) by exact elimination: pinned rows and columns are replaced by identity.

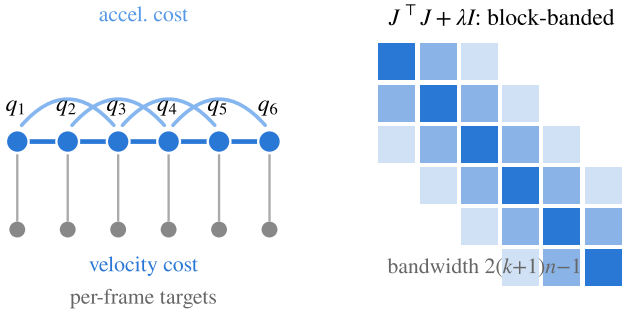

Fig. 2. **Left:** per-frame costs (targets, limits) touch one frame; temporal costs couple $k{+}1$ consecutive frames (velocity $k{=}1$, acceleration $k{=}2$). **Right:** the resulting Gauss–Newton system is block-banded; it is assembled directly in banded form and never materialized as a global Jacobian.

IV. Banded Linear Backends

Higher-order stencils are *reblocked*: grouping k consecutive frames into a superblock of size $m = kn$ turns any banded system with bandwidth $2(k{+}1)n - 1$ into a block-*tridiagonal* system of $N = \lceil T/k \rceil$ superblocks, so all backends operate on one canonical structure. We solve $Ax = b$ with A symmetric positive definite (SPD), block tridiagonal with diagonal blocks D_i and sub-diagonal blocks L_i, batched over B trajectories.

A. Batched block-tridiagonal Cholesky (`direct`)

A block Thomas algorithm runs inside a single tile kernel: one CUDA thread block per trajectory factorizes $D_i - W_i W_i^\top = C_i C_i^\top$ sequentially over superblocks using `tile_cholesky`, `tile_lower_solve` and `tile_matmul`, with forward and backward substitution in the same kernel. The factorization is exact and its sequential-in-time cost is hidden by batch parallelism—$B{=}64$ trajectories cost approximately the same as one—but a single long trajectory leaves the GPU idle: sequential depth is $O(N)$.

B. SPIKE-style parallel-in-time direct solve (`spike`)

To parallelize a single long horizon we adopt a SPIKE-style partitioning [15]: the superblock chain is split into P interior partitions separated by $P{-}1$ single-superblock *interfaces*. Each interior A_p is factorized independently (one thread block per partition per trajectory) by the same block Thomas recursion, simultaneously solving three right-hand sides: the local solution $y_p = A_p^{-1} b_p$ and the two spike columns $U_p = A_p^{-1}\big(e_1 \otimes C_p^L\big)$ and $V_p = A_p^{-1}\big(e_{L_p} \otimes C_p^{R^\top}\big)$ coupling it to its neighboring interfaces. Substituting the interior solutions into the interface rows yields a reduced block-tridiagonal system on the interface unknowns $x_{\sigma_1}, \ldots, x_{\sigma_{P-1}}$ with

$$\hat{D}_j = D_{\sigma_j} - C_j^L V_j^{\text{last}} - C_j^{R^\top} U_{j+1}^{\text{first}}, \qquad \hat{L}_j = -\,C_j^L\, U_j^{\text{last}}, \tag{2}$$

which is exactly the Schur complement of the SPD system onto the interface blocks and is therefore itself SPD—unlike the classic SPIKE reduced system, which is nonsymmetric in general. Consequently the reduced solve reuses the *same* sequential block-Thomas Cholesky kernel over only $P-1$ blocks, and every factorization in the pipeline remains a Cholesky. Interior recovery $x_p = y_p - U_p x_{\sigma_{p-1}} - V_p x_{\sigma_p}$ is embarrassingly parallel over all blocks. Sequential depth drops from $O(N)$ to $O(N/P) + O(P)$; the solve remains exact—it agrees with `direct` to $\sim 10^{-7}$ per iteration and produces digit-identical LM traces in practice (Fig. 6).

C. Block-Jacobi preconditioned conjugate gradients (`cg`)

The banded Hessian is stored as a fixed-topology BSR matrix whose values are rewritten in place each iteration; a batched conjugate-gradient solver runs with per-trajectory offsets and a block-Jacobi preconditioner (per-frame $n \times n$ diagonal blocks inverted with tiled Cholesky solves), warm started across LM iterations. The solve is inexact but parallel over frames; its device-side termination check keeps it CUDA-graph capturable.

V. Experiments

A. Protocol and timing taxonomy

All measurements: NVIDIA RTX PRO 6000 Blackwell workstation GPU (driver 595.84), fp32 throughout, fixed random seeds. We report three timing regimes and never mix them: *(i) compile* — one-time Warp module load and tile-kernel JIT per (DoF, residual, bandwidth) specialization for us; XLA `jax.jit` compilation for PyRoKi/jaxls; warm-up and CUDA-graph capture for cuRobo; *(ii) cold* — first optimized solve after compilation, no graph replay; *(iii) steady state* — CUDA-graph replay for us and cuRobo, re-invocation of the compiled XLA callable for jaxls; median of 10 runs after 3 warm-ups. Published cuRobo and PyRoKi numbers correspond to regime *(iii)*, so all cross-library comparisons below are steady state. Baseline versions: cuRobo v0.7.8 (v1 API, source build, CUDA graphs enabled, its own benchmark configuration; pinned for comparability with PyRoKi's published protocol—the cuRoboV2 rewrite [12] changes conventions and is discussed in Sec. VI) and current PyRoKi/jaxls (JAX 0.10, cuda12 wheels).

B. Single-frame parity

Before making trajectory claims, Table I verifies the underlying single-frame solver is competitive under the cuRobo/PyRoKi benchmark convention: targets from forward kinematics of uniform-random configurations, success = position error $< 5\,\mathrm{mm}$ and orientation error $< 0.05\,\mathrm{rad}$, errors over successful solves. The Newton solver (64 Roberts-sequence seeds, 16 LM iterations, analytic Jacobians) is 5–$16\times$ faster than both baselines at every batch size on the identical FR3 model (PyRoKi) and cuRobo's packaged Panda, with $\geq 99.5\%$ success; relative standings match numbers previously measured on an RTX 4090, supporting hardware-independence of the comparison.

TABLE I

Single-frame IK, steady state (ms, median of 10). Franka: one 6-DoF pose target. H1 (19 DoF): four simultaneous pose targets (both elbows and ankles), error = max over targets. Success: 5 mm/0.05 rad.

Robot	Solver	b=1	b=10^2	b=10^3	succ. %
Franka FR3	cuRobo v0.7.8	5.55	6.59	17.1	99.9–100
	PyRoKi (IK-Beam)	7.51	13.7	27.7	100
	Newton	**0.71**	**0.87**	**3.17**	100
Unitree H1*	PyRoKi (IK-Beam)	12.5	18.6	229	70.7–100
	Newton	**0.85**	**0.97**	**24.9**	99.5–100

*H1 batch columns are $b \in \{1, 10, 10^3\}$; cuRobo targets manipulators.

TABLE II

Whole-trajectory solve, FR3, T=120 @ 30 Hz, B=4, 16 LM iterations. $e_{\max}$: max end-effector deviation; $|\ddot{q}|_{\max}$, $\overline{|\dddot{q}|}$: peak acceleration and mean jerk.

Method	cold [ms]	steady [ms]	$e_{\max}$ [mm]	$\lvert\ddot{q}\rvert_{\max}$ [rad/s^2]	$\overline{\lvert\dddot{q}\rvert}$ [rad/s^3]
Newton direct	17.9	6.1	11.4	13.4	8.5
Newton SPIKE	18.1	4.9	11.4	13.4	8.5
Newton BJ-PCG	49.9	2.7	11.3	13.4	8.6
jaxls/PyRoKi (same problem)	—	217	11.4	13.4	8.5
sequential per-frame IK	715	—	**0.0**	39.2	30.5

cuRobo motion-gen (start→goal, 32 knots; different problem): 56.5 / 54.7 ms.

C. Trajectory solve: identical problem, three ways

The shared task tracks a pick-and-place end-effector path on the FR3: T=120 frames at 30 Hz, B=4 trajectories with laterally offset waypoints, orientation held, frame 0 pinned, identical weights across implementations (position 1.0, rotation 0.5, limits 10, velocity smoothness 0.01, acceleration 5×10^{-4}, rest pose 10^{-3}). Table II compares our three backends against (a) the strongest sequential baseline—the per-frame solver, warm started and batched across the four trajectories, 16 iterations per frame, the same pattern as cuRoboV2's MotionRetargeter—and (b) PyRoKi/jaxls solving the *same* joint problem (per-frame variables, stencil costs, sparse LM with block-Jacobi-preconditioned CG). Newton and jaxls converge to the same solution ($11.36\,\mathrm{mm}$ maximum end-effector deviation at path corners for both—the intended smoothness/tracking trade—and identical smoothness statistics to three digits): the comparison isolates solver efficiency, not solution quality. The joint solve is $40\times$ faster cold than sequential IK and reaches 2.7–6.1 ms steady state; jaxls needs $217\,\mathrm{ms}$ for the identical problem. Sequential IK tracks targets exactly—greedy per-frame solves never trade tracking for smoothness—but at $2.9\times$ the peak acceleration and $3.6\times$ the mean jerk. For context, cuRobo's motion generation (start→goal on 32 B-spline knots with retiming, collision off) takes $56.5\,\mathrm{ms}$ single / $54.7\,\mathrm{ms}$ for a batch of four—a different, easier problem than full-path tracking, listed for scale only.

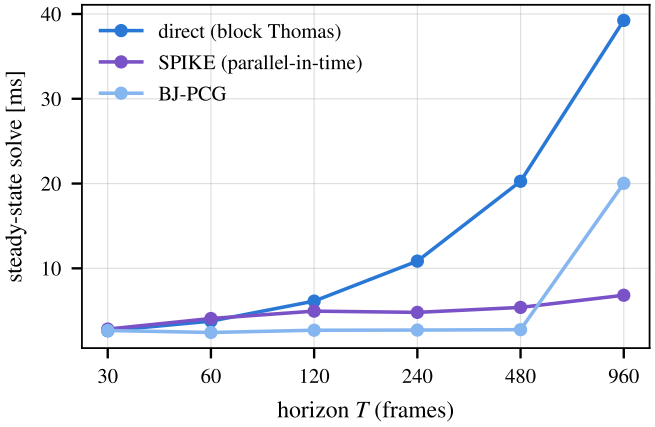

Fig. 3. Steady-state solve time vs. horizon T (FR3, B=4, 16 LM iterations). The parallel-in-time SPIKE backend removes the linear-in-T latency of the sequential direct sweep while keeping the solve exact.

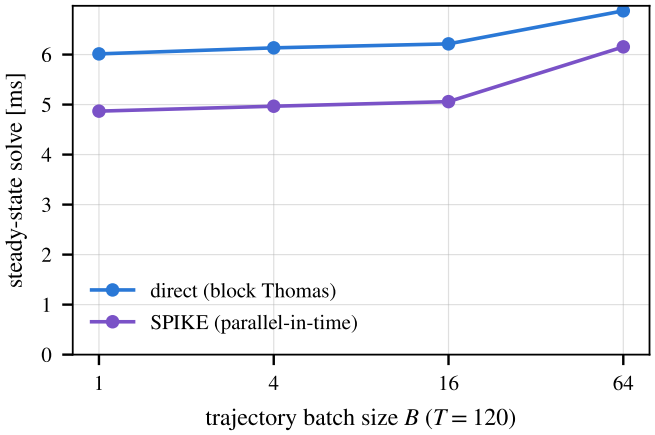

Fig. 4. Steady-state solve time vs. trajectory batch size B at T=120. One CUDA thread block per trajectory (direct) or per partition (SPIKE) makes batching nearly free.

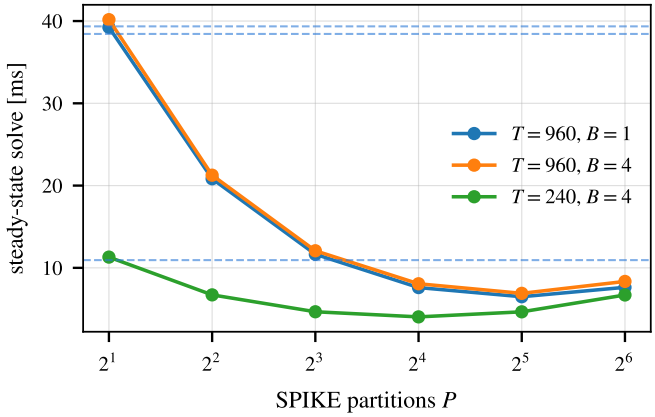

Fig. 5. SPIKE partition-count ablation (steady state; dashed lines: the sequential `direct` baseline for each configuration). P=2 matches the sequential sweep; the optimum balances interior factorization against the $O(P)$ reduced solve.

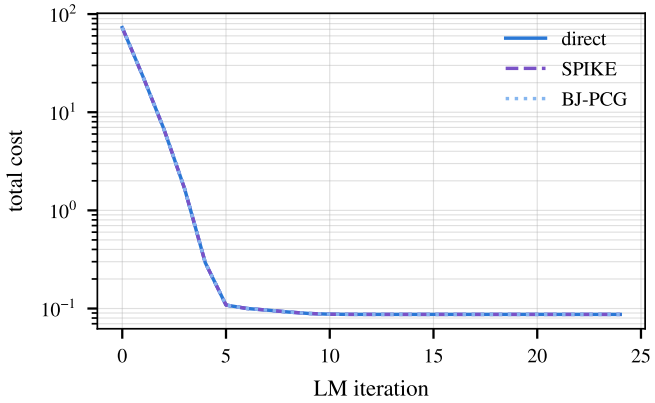

Fig. 6. Total cost vs. LM iteration on the task of Table II. The two exact backends (direct, SPIKE) coincide to machine precision; block-Jacobi PCG tracks them closely.

D. Scaling and backend crossover

Fig. 3 sweeps the horizon at B=4. The sequential `direct` sweep is linear in T (2.6 ms at T=30 to 39.2 ms at T=960); `spike` stays nearly flat (6.8 ms at T=960, 5.8× faster than `direct`) while remaining exact, and is the fastest backend at long horizons; BJ-PCG's data-dependent inner loop lands between the two (20 ms at T=960). Fig. 4 sweeps the batch at T=120: both direct backends are essentially flat from 1 to 64 trajectories (6.0 → 6.9 ms and 4.9 → 6.2 ms)—batch parallelism is nearly free, so backend choice is governed by horizon length and exactness requirements, not batch size. Fig. 5 ablates the SPIKE partition count: P=2 degenerates to the sequential sweep, throughput saturates around $P \approx$ 16–32 where interior work and the $O(P)$ reduced solve balance, and the default heuristic ($P \approx N/16$) lands near the optimum. Fig. 6 confirms the exactness hierarchy: `direct` and `spike` produce digit-identical LM cost traces; BJ-PCG deviates only in the fourth significant digit.

Coverage across batch and horizon. Fig. 7 completes the timing picture over the (T, B) grid for both robots at equal 16-iteration budgets on identical FK-feasible targets. On the arm, SPIKE is the fastest backend everywhere beyond the smallest problems (the sequential direct sweep edges it by <10% at T=120 and at B=256 for $T \leq$960, where batch parallelism already saturates the GPU) — up to 12.6× over the direct sweep and 233× over the warm-started per-frame loop at (T=3840, B=1) — and an equal-iteration jaxls/PyRoKi run of the same problem trails even the per-frame loop (239× vs. SPIKE at T=1920; the equal-*quality* protocol of Table II gives the smaller 12×, since jaxls converges in fewer outer iterations than it is granted here). On the humanoid, where SPIKE's interior exceeds the tile budget, CG leads at small batches (88× over per-frame at T=960, B=1; 43× over direct at T=3840), while the direct sweep becomes competitive once batch parallelism hides its serial depth (1.2× *faster* than CG at T=240, B=256). The per-frame loop is flat in B: batching rescues most of its throughput gap (to within 1.9× of CG at B=64, the corpus regime) but not its greedy quality.

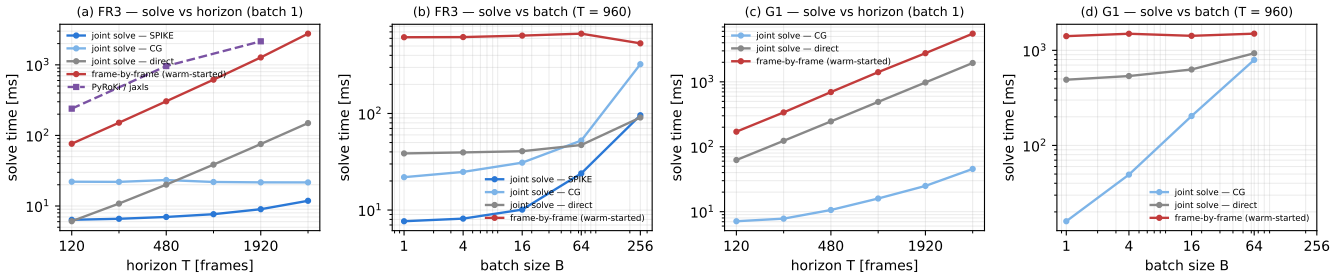

Fig. 7. Core timing grid: steady-state whole-trajectory solve time (16 LM iterations, CUDA-graph replay, median of 5; identical synthetic FK-feasible targets across all methods). **(a, b)** FR3. **(c, d)** G1 — 35 DoF with velocity-order smoothing, so the reblocked direct kernels fit the tile budget; SPIKE's three-RHS interior does not at m=35 (Sec. VI). PyRoKi/jaxls runs the identical problem at the same iteration budget (FR3 only: the G1 task's floating base has no off-the-shelf jaxls variable).

E. Case study: humanoid motion retargeting

We retarget SOMA motion-capture clips to a floating-base 29-DoF Unitree G1 using the production configuration of an internal retargeting pipeline (14 weighted body targets from scaled human keypoints, native 120 Hz); the free-joint base makes the body-centered tangent convention of Sec. III load-bearing. The baseline is that pipeline's own production pattern: per-frame IK, warm started, 24 iterations per frame (the cuRoboV2 MotionRetargeter pattern). The joint solve adds velocity smoothness on the actuated joints—the mocap root motion is signal, so the base is excluded via per-DoF weights—and solves each full clip in one shot.

On a fast dance clip (690 frames, 5.8 s), sequential IK tracks the (feasible) targets to 3.6 mm but with peak joint acceleration of 9.1×10^2 rad/s^2—visible jitter that production pipelines suppress with post-processing. The joint solve trades 13 mm mean key-target error for a 14× reduction in peak acceleration, in 0.2 s for the whole clip versus 29 s sequentially. A straight-leg walking clip (1088 frames) is a stress case: the human-proportioned foot targets exceed the G1's leg reach (hip-to-foot distances up to 0.79 m versus 0.766 m at full extension), and greedy per-frame IK chatters at the workspace boundary (peak $|\ddot{q}| \approx 5.6 \times 10^2$ rad/s^2 versus the joint solve's 38, a 15× gap; 46 s per clip). Both sides here benefit from the body-centered base tangents of Sec. III: the same convention that keeps the joint solve convergent far from the origin also damps the per-frame baseline's boundary oscillation, and the smoothness gap that remains is structural greediness, not a Jacobian artifact. On this 35-DoF problem the horizon-parallel backends matter: at 1088 frames the sequential direct sweep takes 1.1 s versus 0.21 s for the conjugate-gradient path. Acceleration-order smoothing at 35 DoFs exceeds the shared-memory budget of the reblocked direct kernels ($m = 2n = 70$), so the direct backend runs velocity-order smoothing and the CG path carries the acceleration term; the SPIKE interior kernel, which holds three concurrent right-hand sides, hits the same budget already at $m = 35$—instances of the limitation discussed in Sec. VI.

Corpus scale: BONES-SEED: The retargeting pipeline scales to the full BONES-SEED corpus [33] (288 h of motion; 142,220 clips at 120 Hz, with production G1 retargets shipped for reference): streaming BVH preprocessing of all 124.5M frames takes 21 min on 14 CPU cores (a vectorized, parity-tested rewrite of the production loader), and the joint solve of the whole corpus takes 2.7 h on one GPU in length-bucketed batches of 64 clips — versus ∼15 h projected for the production-style batched per-frame loop at its measured rate on the same GPU. Quality against the shipped production data is two-sided (Table III): the joint solve is markedly smoother (5.7× lower peak acceleration, 2.2× lower jerk), while the production loop's per-frame feet-stabilizer post-pass tracks tighter and hides ground contact errors that its pipeline (which has no contact handling) would otherwise exhibit. Adding the ported objectives of Sec. III-C inside the solve — ground-plane penalties on foot points and whole-body capsule surfaces, contact-gated stance pins, and a foot-skate lock, with contact labeled from the source — drops both sole and body-surface penetration below the production data's own rates, closes the tracking gap to half a millimeter, and retains a 2–3× smoothness advantage. The inexact inner solve is not a quality liability: re-solving the evaluation subset with an exact direct factorization moves tracking and contact metrics by under 3% (n=14,222 paired clips), with only the smoothness statistics trading (lower mean jerk for higher peak acceleration).

TABLE III

BONES-SEED CORPUS VS THE SHIPPED PRODUCTION RETARGETS (EVERY 10TH CLIP, n=14,222; MEANS; SOURCE-LABELED CONTACTS). JOINT SOLVE: RETUNED PLAIN CONFIG; + PORTED OBJ.: PLANE, CAPSULE, STANCE-PIN, FOOT-SKATE, REST-POSE. SOLE PEN. IS ANKLE-SOLE DEPTH; BODY PEN. IS WHOLE-BODY CAPSULE-SURFACE DEPTH (>1 CM).

	joint solve	+ ported obj.	production
key-effector err [mm]	10.7	1.9	1.4
foot slide [cm/s]	7.7	2.3	2.2
sole pen. frames [%]	11.7	1.3	3.7
body pen. frames [%]	5.1	3.9	5.2
max joint accel [rad/s^2]	151	270	857
mean joint jerk	473	461	1022

F. Case study: torque- and balance-aware trajectories

Both objectives of Sec. III-C are exercised on a Franka FR3.

TABLE IV

POST-HOC INVERSE-DYNAMICS METRICS, FRANKA FR3 + 3 KG PAYLOAD, 4 S LATERAL CARRY, ARM JOINTS 1–7. LIMIT UTILIZATION IS AGAINST THE PER-JOINT FR3 TORQUE LIMITS (87/87/87/87/12/12/12 N·M).

objectives	$\max \lvert\tau\rvert$ [N·m]	util. [%]	$\overline{\lvert\tau\rvert}$ [N·m]	EE err [mm]
position + rotation only	108.9	125	12.8	0.0
+ vel/accel smoothing	71.9	83	12.7	13.7
+ gravity torque, $w{=}5{\cdot}10^{-4}$	66.4	76	12.1	15.1
+ gravity torque, $w{=}10^{-3}$	53.1	61	11.7	24.6
+ gravity torque, $w{=}2{\cdot}10^{-3}$	49.2	57	10.4	65.4

Gravity torque. The arm carries a 3 kg payload along a fixed 4 s lateral path (position, orientation, limit, reference, and smoothness objectives as in Sec. V-C). Torques are evaluated post hoc with Newton's inverse dynamics, $\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q)$, with $\dot{q}, \ddot{q}$ by central differences, all 120 frames in one batched call on a 120-world replicated model. Table IV and Fig. 8: without smoothing, per-frame-style tracking exceeds the shoulder's 87 N·m limit (125%); smoothing brings it to 83%; the gravity-torque objective at $w{=}10^{-3}$ re-postures the arm to 61% for a 25 mm tracking concession; mean torque drops a more modest 8–18% (and mechanical energy stays roughly flat — the payload's potential-energy profile is fixed by the path). The re-posturing is qualitative and visible: the arm chooses a flatter, gravity-friendly configuration over the whole path (video on the project page).

Apparent gravity, validated in simulation. The waiter objective is validated by running the loop it approximates: the IK output drives a PD-controlled FR3 (gravity-compensated actuators, velocity feedforward, MuJoCo solver with elliptic friction cone) carrying a 13 cm-half-width plate with a *free ball* resting on it, through a fast lateral dash and reversal. A level-plate orientation objective loses the ball at the first acceleration peak — a rolling ball lags any level dash by $5/7$ of the plate's motion, so this failure is structural, not a friction-margin issue — while the apparent-gravity objective banks the plate into the motion and keeps the ball aboard for the whole run, with a peak offset of 1.4 cm (Fig. 9). The banking is planned by the trajectory solve, not by feedback: the simulation replays the IK joint targets open loop.

G. Case study: whole-program toolpath solves

Prescribed task-space paths in manufacturing — laser marking and texturing, robotic additive manufacturing, spray coating, ultrasonic scan coverage — are the extreme batch-one regime: the entire program exists before the robot moves, horizons reach millions of frames, and the arm is low-DoF, which is exactly where the parallel-in-time backend applies ($m = kn$ within the tile budget, Sec. VI). We engrave a raster image onto a domed panel with an FR3: each pixel sample is one frame of a serpentine toolpath, and the tool is axisymmetric, so an axis-alignment objective points the flange axis along the surface normal and leaves the spin free as redundancy (position, joint-limit, velocity/acceleration smoothness, and reference objectives as before). A 1152×912 raster is a $T{=}1{,}050{,}624$-frame trajectory — 2.4 h of execution at 120 Hz — solved as one nonlinear problem in **1.1 s** with SPIKE (32 LM iterations, 9 GiB; CG 2.9 s; the sequential direct sweep 104 s), versus 11.7 min for the warm-started per-frame loop at 16 iterations per frame (Fig. 10a). An equivalent jaxls/PyRoKi formulation of the same problem (same weights and iteration budget, custom axis cost) solves the million-frame program in 81 s after a 91 s XLA compile — $73\times$ the SPIKE time; at this scale jaxls's iterative inner solve compounds against the parallel-in-time exact factorization. All solutions track at 0.01 mm mean; the difference is dynamics and latency. Greedy per-frame IK carries a $5\times$ acceleration spike into every serpentine reversal and, more damagingly, converts the *image content itself* into joint-jerk ripple — the engraved figure is visible in its vibration-excitation map, exactly where marking quality matters — while the joint solve pre-shapes reversals and confines excitation to the off-image margins (Fig. 10b,c). Whole-program latency is the workflow win: re-posing the workpiece and re-solving the entire 2.4 h program takes seconds, turning CAM verification into an interactive loop.

H. Compilation cost

One-time costs, measured: Warp tile-kernel JIT is 30–60 s per (DoF, residual-dimension, bandwidth) specialization and is cached on disk across runs and across horizon/batch sizes—T and B are runtime dimensions, so the sweeps of Figs. 3–4 reuse one compilation. jaxls/XLA compiles per shape: 3.6–10.3 s per (robot, batch) single-frame graph and 11.3 s for the $T{=}120$ trajectory problem, re-triggered by any shape change. cuRobo absorbs warm-up and CUDA-graph capture per problem shape (unreported in its benchmarks; seconds in practice).

VI. LIMITATIONS AND FUTURE WORK

Superblock shared-memory cap. Reblocking makes higher-order stencils tridiagonal at block size $m = kn$; tile kernels hold several $m \times m$ tiles in shared memory, which bounds $kn \lesssim 64$–96 (fp32) on current hardware. High-DoF humanoids with acceleration or jerk smoothing therefore fall back to the CG path (Sec. V-E), and the SPIKE interior kernel's three simultaneous right-hand sides lower its cap by roughly $3\times$ relative to the plain sweep; a multi-pass interior factorization and a non-reblocked banded kernel would remove both restrictions. **Free-joint tangents.** Root free-joint tangents are body-centered (rotation pivots about the base's own anchor), so convergence is invariant to where a trajectory sits in the world; non-root free joints keep the parent-anchor convention. **Baselines.** Our cuRobo comparisons use v0.7.8 (v1 API) for continuity with PyRoKi's published protocol; benchmarking against cuRoboV2 [12]—in particular its whole-body G1 IK and its frame-by-frame MotionRetargeter on identical clips—is the natural next experiment, as is GMR [3] for the retargeting study. **Scope.** Collision terms cover world-plane points and capsule surfaces plus self-collision capsule pairs (Sec. III-C); general environment

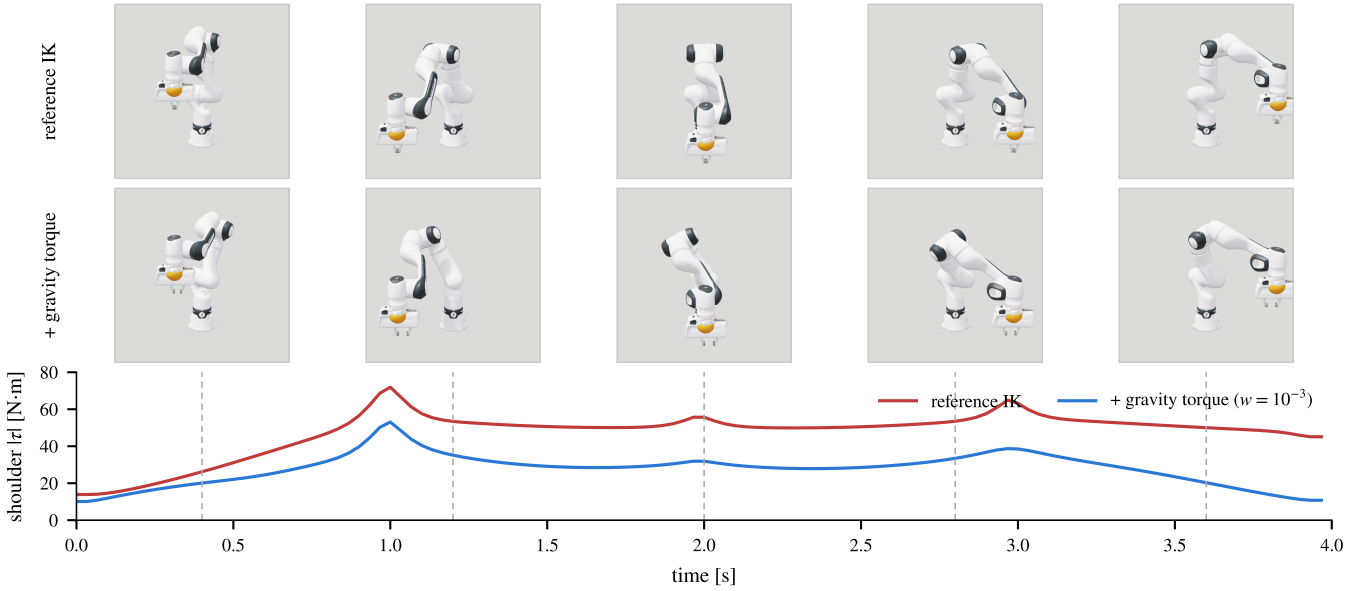

Fig. 8. Gravity-torque objective: Franka FR3, 3 kg payload, same end-effector path. Stills are placed at their time coordinates (dashed guides) above the shoulder torque from the inverse-dynamics check. The reference solve (top) holds the payload high with a winged elbow; the gravity-torque solve (bottom) settles into a flatter, gravity-friendly posture over the whole path, cutting the torque peaks (Table IV).

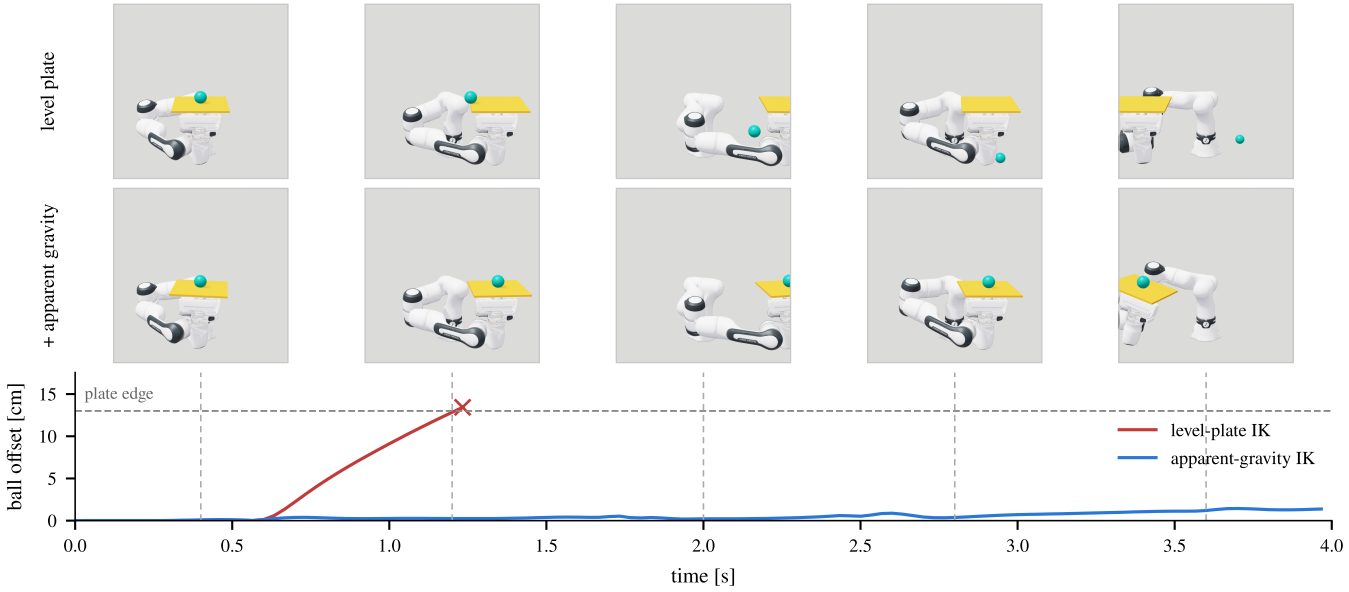

Fig. 9. Ball-on-plate validation: the IK output drives a PD-controlled forward simulation carrying a free ball. Stills are placed at their time coordinates (dashed guides) above the simulated ball offset (× marks the ball leaving the plate). The level-plate solve (top) sheds the ball at the first acceleration peak; the apparent-gravity solve (bottom) banks the plate up to ∼15° into the accelerations and the ball rides out the whole run at ≤1.4 cm offset.

geometry and swept variants are future work, though the off-diagonal coupling they need is already exercised by the apparent-gravity objective. Hard constraints currently enter as weighted penalties; an augmented-Lagrangian outer loop is mechanical future work. **Dynamics objectives.** The gravity-torque objective captures only the static term of the manipulator equation; its Hessian is Gauss–Newton-approximate between rotational DoFs sharing one ball or D6 joint. Manipulator torque *limits* are generous — a stock FR3 runs our 3 kg carry at 83% of its binding shoulder limit with smoothing alone — so the honest motivation is effort and heat, not limit avoidance, and the win grows with payload. The apparent-gravity objective penalizes tangential force quadratically without a unilateral-contact or friction-cone constraint, and its simulation validation is open loop: the PD-tracked plate realizes the planned banking only approximately. Torque with the full $M(q)\ddot{q}$ term is a stencil-2 objective whose dominant block is $M(q)\, s_j/\Delta t^2$; the assembly supports it unchanged.

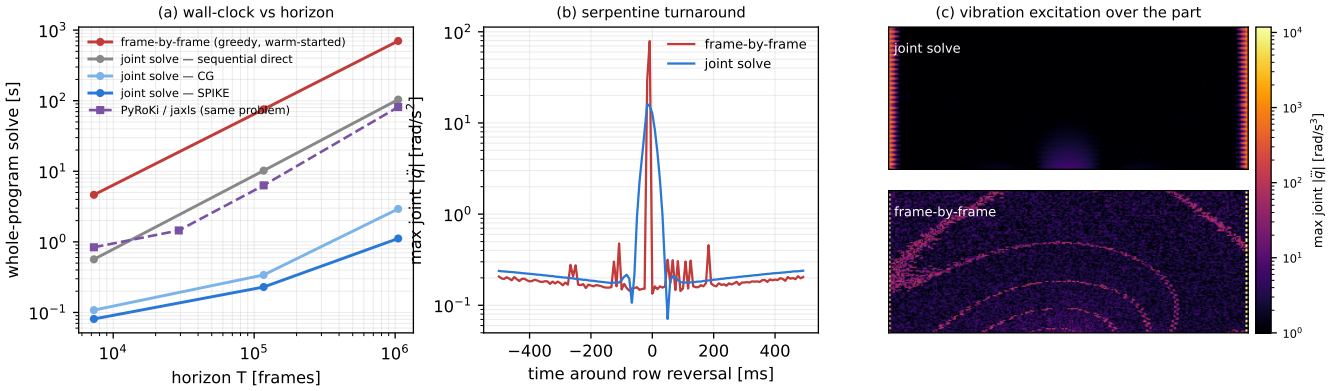

Fig. 10. Whole-program engraving on a 7-DoF FR3 (raster image over a domed panel; one frame per pixel sample). **(a)** Whole-program solve time vs. horizon, batch $= 1$: at T=1.05M frames the banded solves are 6.8× (direct) to 626× (SPIKE) faster than the sequential warm-started per-frame loop; an equivalent jaxls/PyRoKi formulation (dashed) lands at 73× the SPIKE time. **(b)** Peak joint acceleration through one serpentine row reversal. **(c)** Vibration excitation (max joint jerk per raster sample, log scale) over the part: the per-frame solve (bottom) reproduces the engraved image as jerk ripple; the joint solve (top) confines excitation to the off-image turnaround margins.

VII. Conclusion

Whole-trajectory IK is a block-banded nonlinear least-squares problem, and on modern GPUs the banded system can be solved directly, in batch, and—with a SPIKE-style symmetric Schur reduction—in parallel across time, while remaining CUDA-graph capturable end to end. Against the prevailing frame-by-frame pattern this yields order-of-magnitude faster solves with substantially smoother trajectories; against a state-of-the-art GPU nonlinear-least-squares framework solving the identical problem it is an order of magnitude faster at equal solution quality. The batched banded solve—the canonical kernel of trajectory optimization, Kalman smoothing, and spline fitting—belongs in GPU tile-programming libraries.

References

[1] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox, "curobo: Parallelized collision-free minimum-jerk robot motion generation," 2023.

[2] C. M. Kim, B. Yi, H. Choi, Y. Ma, K. Goldberg, and A. Kanazawa, "Pyroki: A modular toolkit for robot kinematic optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025, arXiv:2505.03728.

[3] 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, arXiv:2510.02252.

[4] T. He, Z. Luo, W. Xiao, C. Zhang, K. Kitani, C. Liu, and G. Shi, "Learning human-to-humanoid real-time whole-body teleoperation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 8944–8951.

[5] T. He, Z. Luo, X. He, W. Xiao, C. Zhang, W. Zhang, K. M. Kitani, C. Liu, and G. Shi, "OmniH2O: Universal and dexterous human-to-humanoid whole-body teleoperation and learning," in *Conference on Robot Learning (CoRL)*, 2024, pp. 1516–1540.

[6] Z. Luo, J. Cao, A. Winkler, K. Kitani, and W. Xu, "Perpetual humanoid control for real-time simulated avatars," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[7] M. Toussaint, "Newton methods for k-order markov constrained motion problems," 2014.

[8] 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, pp. 2536–2542.

[9] W. Jallet, A. Bambade, E. Arlaud, S. El-Kazdadi, N. Mansard, and J. Carpentier, "ProxDDP: Proximal constrained trajectory optimization," *IEEE Transactions on Robotics*, vol. 41, pp. 2605–2624, 2025.

[10] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs," in *Robotics: Science and Systems (RSS)*, 2016.

[11] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

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

[13] E. Adabag, M. Atal, W. Gerard, and B. Plancher, "MPCGPU: Real-time nonlinear model predictive control through preconditioned conjugate gradient on the GPU," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 9787–9794.

[14] M. Macklin, "Warp: A high-performance Python framework for GPU simulation and graphics," https://github.com/nvidia/warp, 2022.

[15] E. Polizzi and A. H. Sameh, "A parallel hybrid banded system solver: The SPIKE algorithm," *Parallel Computing*, vol. 32, no. 2, pp. 177–194, 2006.

[16] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[17] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[18] NVIDIA Isaac, "cumotion: GPU-accelerated motion generation for robotics," https://github.com/nvidia-isaac/cumotion, 2026.

[19] B. Yi, M. A. Lee, A. Kloss, R. Martín-Martín, and J. Bohg, "Differentiable factor graph optimization for learning smoothers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1339–1345.

[20] B. Yi, "jaxls: Sparse, constrained, and non-Euclidean least squares in JAX," https://github.com/brentyi/jaxls, 2025.

[21] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9–10, pp. 1164–1193, 2013.

[22] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574.

[23] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with MuJoCo," 2022.

[24] J. Schulman, Y. Duan, J. Ho, A. 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*, vol. 33, no. 9, pp. 1251–1270, 2014.

[25] A. Du, E. Adabag, G. Bravo-Palacios, and B. Plancher, "GATO: GPU-accelerated and batched trajectory optimization for scalable edge model predictive control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2026, arXiv:2510.07625.

[26] S. H. Jeon, S. Hong, H. J. Lee, C. Khazoom, and S. Kim, "CusADi: A GPU parallelization framework for symbolic expressions and optimal control," *IEEE Robotics and Automation Letters*, vol. 10, no. 2, pp. 899–906, 2025.

[27] Y. Zou, Z. Zhang, M. Robic, and F. Jiang, "Parallel-in-time nonlinear optimal control via GPU-native sequential convex programming," 2026.

[28] 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, pp. 8749–8756.

[29] B. L. Buzbee, G. H. Golub, and C. W. Nielson, "On direct methods for solving Poisson's equations," *SIAM Journal on Numerical Analysis*, vol. 7, no. 4, pp. 627–656, 1970.

[30] E. Polizzi and A. H. Sameh, "SPIKE: A parallel environment for solving banded linear systems," *Computers & Fluids*, vol. 36, no. 1, pp. 113–120, 2007.

[31] S. Särkkä and Á. F. García-Fernández, "Temporal parallelization of Bayesian smoothers," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 299–306, 2021.

[32] ——, "Temporal parallelization of dynamic programming and linear quadratic control," *IEEE Transactions on Automatic Control*, vol. 68, no. 2, pp. 851–866, 2023.

[33] Bones Studio and NVIDIA, "BONES-SEED: Humanoid motion dataset," https://huggingface.co/datasets/bones-studio/seed, 2026.