2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Differentiable Particle Optimization for Fast Sequential Manipulation

Lucas Chen, Shrutheesh R. Iyer, and Zachary Kingston

***Abstract*— Sequential robot manipulation tasks require finding collision-free trajectories that satisfy geometric constraints across multiple object interactions in potentially high-dimensional configuration spaces. Solving these problems in real-time and at large scales has remained out of reach due to computational requirements. Recently, GPU-based acceleration has shown promising results, but prior methods achieve limited performance due to CPU-GPU data transfer overhead and complex logic that prevents full hardware utilization. To this end, we present SPaSM (Sampling Particle optimization for Sequential Manipulation), a fully GPU-parallelized framework that compiles constraint evaluation, sampling, and gradient-based optimization into optimized CUDA kernels for end-to-end trajectory optimization without CPU coordination. The method consists of a two-stage particle optimization strategy: first solving placement constraints through massively parallel sampling, then lifting solutions to full trajectory optimization in joint space. Unlike hierarchical approaches, SPaSM jointly optimizes object placements and robot trajectories to handle scenarios where motion feasibility constrains placement options. Experimental evaluation on challenging benchmarks demonstrates solution times in the realm of milliseconds with a 100% success rate; a $4000\times$ speedup compared to existing approaches. Code and examples are available at commalab.org/papers/spasm.**

I. Introduction

Robotic manipulation tasks require executing sequences of coordinated motions where each action fundamentally depends on the successful completion of preceding actions. These dependencies arise from geometric constraints imposed by object grasping configurations, kinematic limitations of robot arms, and obstacles that restrict feasible motion paths. The coupling between action grounding (i.e., placement of the objects) and continuous motion planning creates a computationally challenging problem where local failures can cascade through the entire action sequence, resulting in failure to find feasible motion.

Recent advances in GPU-accelerated motion planning and manipulation planning such as cuRobo [1] and cuTAMP [2] have demonstrated significant computational improvements by parallelizing operations across thousands of concurrent threads. These methods leverage GPU hardware to accelerate computation. However, there are still bounding CPU-based coordination steps which incur overhead from CPU-GPU data transfers. The computational bottleneck has shifted from individual constraint evaluations to the coordination and data movement between processing units.

To address this, we present SPaSM (Sampling Particle Optimization for Sequential Manipulation), a GPU-parallel framework that performs end-to-end optimization of manipulation trajectories entirely on GPU hardware. We focus on the problem of finding collision-free feasible motion sequences that satisfy a predetermined sequence of manipulation actions. Given a fixed action skeleton specifying the order of pick-and-place operations, our approach jointly optimizes object placements and robot trajectories to minimize execution cost while satisfying all geometric and kinematic constraints. The resulting problem structure admits massive parallelization across both candidate solutions and optimization iterations, with zero CPU coordination overhead. Our method uses a two-stage approach: first optimizing object placement configurations through parallel particle optimization, then lifting successful placements to full trajectory optimization in joint space. Unlike previous approaches, our second phase performs joint optimization of robot motion and object positions to handle scenarios where motion feasibility significantly constrains placement options. The entire optimization—including sampling, constraint evaluation, gradient computation, and trajectory refinement—executes as compiled CUDA kernels without CPU intervention, leveraging JAX [3] for efficient implementation. Our implementation is open source[1].

Evaluation on challenging benchmarks demonstrates that SPaSM achieves solution times in *milliseconds* on complex benchmark problems, representing a $4000\times$ speedup compared to cuTAMP (without action skeleton search) while maintaining 100% success rates. Our approach scales effectively and is capable of solving reactive manipulation scenarios where environmental conditions change. These performance gains enable real-time replanning capabilities and open new possibilities for manipulation tasks requiring rapid adaptation to dynamic environments.

II. Related Work

Sequential robot manipulation tasks require coordinated planning across both discrete action sequences and continuous motion trajectories. In this work we focus on finding feasible geometric motions and action groundings (i.e., grasp poses and placement locations) over a given sequence of actions—we assume this sequence is given. The MoveIt Task Constructor [4] addresses similar problems by working from prespecified action sequences to generate executable plans via sampling potential groundings and resolving with motion planning. Related sequential manifold planning approaches [5] tackle sequences of actions specified as manifold constraints. We contrast this assumption against Task and Motion Planning (TAMP) problems, where algorithms must also find the high-level sequence of actions that have

LC, SRI, and ZK are with Department of Computer Science, Purdue University, {chen4007, iyer480, zkingston}@purdue.edu

[1] https://github.com/CoMMALab/SPaSM

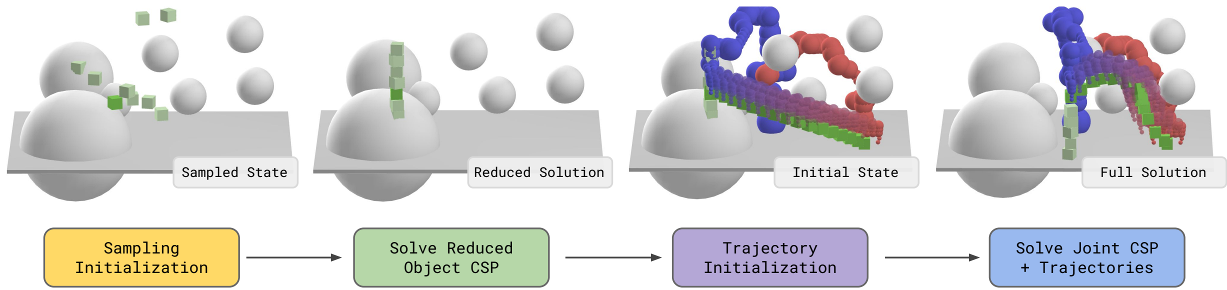


Fig. 1: Overview of our approach, Sampling Particle optimization for Sequential Manipulation (SPaSM), applied to our 10-block tower problem in clutter. SPaSM jointly optimizes over the robot configuration and object states for sequential pick and place tasks using large scale particle optimization on the GPU, solving end-to-end placements and motions for this problem in 12 milliseconds.

corresponding low-level groundings of said actions [6]. However, both classes of problems require solving for satisfying assignments of the grounding variables, constrained by the geometry of the scene and kinematic feasibility. The core challenge is coupling discrete action selection with continuous planning; some approaches tackle this with sampling [7] or optimization [8]. In this work, we assume a valid grounding exists and attempt to find solutions through highly parallel differentiable particle optimization, quickly exploring the potential space of groundings.

Relatedly, trajectory optimization approaches [9] formulate planning as an optimization problem, which naturally captures the complex constraints imposed over long horizon problems. Similar to our approach in formulation is that of logic-geometric programming [10, 11], which also optimizes over both poses of objects and the entire trajectory in a three-layer approach. Optimization-based approaches for grounding leverage gradient information but face challenges with highly nonlinear constraints, requiring good initialization to avoid local minima. We avoid this by leveraging parallelism to sample many initial seeds.

There has also been recent progress in receding horizon sequential planning, using model-predictive control frameworks [12–15]. In contrast, our approach plans an entire trajectory from start-to-finish. It is of interest to adapt our method to the receding horizon case.

A. Parallelization for Planning

Parallel planning has been desirable since the start of the field [16] due to the computational intensity of high-dimensional search problems. Recently, geometric motion planning has been demonstrated to work in the microsecond regime, opening new possibilities for fast and efficient planning [17]. Many parts of planning are embarrassingly parallel [18]. Early GPU implementations focused on specific components such as parallel RRT variants [19] and collision checking acceleration [20]. Recent GPU planners (e.g., [21]) parallelize multiple components of the planner to take advantage of all parts of the hardware. We use JAX [3] to automatically compile our approach into an efficient kernel that can be parallelized.

Trajectory optimization can be accelerated effectively on the GPU [22–25]; recent approaches, such as cuRobo [1], achieve significant speedups for motion planning. Planning formulated as optimal transport problems [26] and Global Tensor Motion Planning [27] exploit tensor operations for massive parallelization. These methods, and others such as PyRoki, leverage JAX for differentiable robot planning [28].

For sequential manipulation specifically, recent probabilistic approaches like STAMP [29] apply parallel simulation and Stein variational gradient descent to explore multiple potential action sequences. cuTAMP [2] represents current state-of-the-art GPU acceleration for long-horizon problems, using differentiable collision checking and constraint satisfaction. To deal with local minima sensitivity, they employ initialization strategies like backtracking and early stopping to eliminate unpromising samples. However, these operations introduce branching operations that require CPU-based coordination to maintain dynamic constraint sets as necessary for solving TAMP problems, creating communication overhead that limits speedups to seconds rather than milliseconds. The hierarchical decomposition between placement optimization and trajectory planning in cuTAMP also prevents joint optimization of interdependent constraints. This also prevents joint optimization of the final trajectory.

Our approach differs fundamentally by eliminating CPU-GPU coordination through end-to-end compiled CUDA kernels, jointly optimizing placement and trajectory feasibility rather than hierarchical decomposition, and achieving millisecond performance. SPaSM also recognizes that motion feasibility can significantly constrain placement options, and thus we integrate optimization across both discrete grounding and continuous trajectory variables.

III. Methodology

We address sequential pick-and-place problems where the geometric properties of all objects, their initial positions, and environmental obstacles are known. This encompasses packing problems (shown in Fig. 5), block stacking tasks (Fig. 1), and other manipulation scenarios requiring multiple coordinated pick-and-place actions. We also assume that the action skeleton—the predetermined sequence of high-level actions—is provided as input. The objective is to determine collision-free object placements that satisfy a constraint function $|f(x)| < \epsilon$, which defines the set of valid intermediate and final configurations, while finding feasible collision-free

motion of the manipulator and potentially the grasped object. We formulate collision constraints as penetration distance penalties between object pairs. Without loss of generality, we represent all objects and the robot as collections of spheres, which enables efficient computation. For manipulator motion, we optimize trajectory cost in terms of distance in the robot's configuration space.

Solving this problem reduces to a Constraint Satisfaction Problem (CSP) [6], requiring optimization over continuous variables subject to multiple constraints, which generates a highly nonlinear optimization landscape. For instance, stacking a tower of N_{obj} blocks requires optimization over approximately $\mathcal{O}(R^d \times N_{\text{timestep}} \times N_{\text{obj}} \times R^{od})$ variables, where R^d represents the robot's degrees of freedom, R^{od} denotes the object state dimensionality, and N_{timestep} indicates the number of waypoints per robot trajectory. To address the high dimensional, non-linear nature of the problem, we use a two-stage optimization approach:

1) **Object Variable CSP**: We solve the reduced CSP for object placement states, obtaining a batch of N candidate solutions that provide collision-free feasible object configurations.
2) **Trajectory Optimization**: We lift these placement solutions into the full configuration space of the robot, jointly optimizing robot configurations, trajectories, and object placements.

Our approach is inspired by the particle optimization proposed by cuTAMP [2]. While cuTAMP only solves the object variable CSP and uses cuRobo to resolve motion, we jointly optimize over entire robot trajectories and object placements. Our approach takes advantage of the known problem structure and uses fused computational kernels for better hardware utilization, resulting in over $4000\times$ faster solve times for the object variable CSP.

A. *Object Variable CSP*

This stage solves only object placement states and computes the final object configurations such that: (i) they maintain collision-free separation from the environment and other objects, and (ii) remain within the problem-specified valid bounding constraints.

Our approach uses parallelized differentiable optimization over batches of particles. Although the optimization variables are continuous, the problem structure often admits only zero-dimensional solution manifolds, making the valid solution set effectively discrete. For example, tightly packing objects within a constrained region typically yields only a finite number of feasible solutions, as demonstrated by the Tetris problem (Fig. 5). The extreme nonlinearity of these constraints presents a fundamental challenge for continuous optimization methods. We leverage large scale parallelism to tackle this problem by increasing the batch size, which empirically seems to be more effective approach for CSP search than other particle initialization techniques

This effectiveness stems from the nature of the parallelism afforded by GPUs, which scale efficiently for simple, branch-free operations such as bulk sampling, rather than complex methods which may result in stalls and coupling between processing units. To improve success rates, we implement a simple rejection sampling procedure: we sample a large number of particles and immediately eliminate high-cost candidates. This approach scales effectively and generalizes well to challenging problems with highly coupled states, as it makes no assumptions regarding state dependencies.

1) Particle Initialization: We perform uniform sampling of N initial states within the problem domain and compute the initial cost for each particle using the constraint function f. We select the M particles with the lowest initial costs for subsequent placement optimization. Following optimization, we verify the existence of successful states and repeat the initialization step if none are found. This corresponds to lines 1–4 in Alg. 1.

This strategy proves effective because initial state cost serves as a reliable (albeit not definite) predictor of optimization success. Fig. 2 demonstrates the cost evolution across all particles for a representative Tetris problem. States with higher initial costs exhibit substantially lower probability of successful convergence. Filtering based on initial cost provides a computationally efficient method for identifying states worthy of optimization investment. The selection of sampling batch size N and optimization batch size M requires careful balancing for performance. While larger sample sizes increase the probability that at least one sample will successfully optimize, excessive sampling wastes computational resources since only a single solution is required. We provide analysis of optimal N and M selections in Fig. 6.

2) Particle Optimization: The selected M particles undergo parallel optimization to determine object placement poses. During optimization, object poses are clamped to ensure they remain within the problem bounding box. The highly nonlinear cost landscape induces multiple local min-

Algorithm 1 GPU-Parallelized Particle Optimization

Require: CSP problem Π, N sampling batch size, M optimization batch size, K_{lin} linear optimization steps, K_{quad} quadratic optimization steps, η_{init} initial learning rate
1: $\mathcal{S} \leftarrow \textsc{SampleUniform}(\Pi, N)$ ▷ Sample initial particles in parallel
2: $\mathcal{C} \leftarrow \textsc{Cost}(\mathcal{S})$ ▷ Evaluate costs for all particles
3: $\mathcal{I} \leftarrow \textsc{ArgSort}(\mathcal{C})[: M]$ ▷ Select indices of M best particles
4: $\mathcal{S}_{\text{opt}} \leftarrow \mathcal{S}[\mathcal{I}]$
5: ▷ **Linear Cost Descent**
6: **for** $k = 1, \ldots, K_{\text{lin}}$ **do**
7: $\quad\eta \leftarrow \eta_{\text{init}} \cdot (1 - k/K_{\text{lin}})$ ▷ Apply learning rate schedule
8: $\quad\mathcal{G} \leftarrow \nabla_{\mathcal{S}_{\text{opt}}} \textsc{Cost}_{\text{linear}}(\mathcal{S}_{\text{opt}})$
9: $\quad\mathcal{S}_{\text{opt}} \leftarrow \mathcal{S}_{\text{opt}} - \eta \cdot \mathcal{G}$ ▷ Perform gradient descent step
10: **end for**
11: ▷ **Quadratic Cost Refinement**
12: **for** $k = 1, \ldots, K_{\text{quad}}$ **do**
13: $\quad\mathcal{G}_{\text{quad}} \leftarrow \nabla_{\mathcal{S}_{\text{opt}}} \textsc{Cost}_{\text{quadratic}}(\mathcal{S}_{\text{opt}})$
14: $\quad\mathcal{S}_{\text{opt}} \leftarrow \mathcal{S}_{\text{opt}} - \alpha \cdot \mathcal{G}_{\text{quad}}$ ▷ Refine with small fixed learning rate α
15: **end for**
16:
17: $\mathcal{S}_{\text{satisfying}} \leftarrow \textsc{GetSatisfyingParticles}(\mathcal{S}_{\text{opt}})$
18: **if** $\mathcal{S}_{\text{satisfying}} \neq \emptyset$ **then**
19: $\quad\mathcal{C}_{\text{final}} \leftarrow \textsc{Cost}(\mathcal{S}_{\text{satisfying}})$ ▷ Re-evaluate final costs
20: $\quad I_{\text{best}} \leftarrow \textsc{ArgSort}(\mathcal{C}_{\text{final}})[: P]$ ▷ Return the best P solutions
21: $\quad$**return** $\mathcal{S}_{\text{satisfying}}[I_{\text{best}}]$
22: **end if**
23: **return** FAILURE

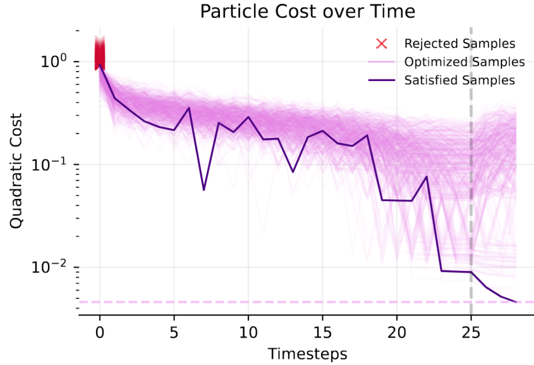


Fig. 2: Cost evolution over 30 optimization steps for the 5-block tetris problem, demonstrating the effectiveness of our optimization strategy. Satisfying states are frequently (though not universally) identified by low initial costs prior to optimization. Pink trajectories represent samples not rejected in the initial step, while red particles represent samples rejected early due to their poor cost. Rejected samples rarely converge to valid solutions. At step 25, the transition to quadratic costs further separates satisfying states from those with low costs but cannot converge.

ima, and naïve cost descent may overlook nearby viable states. To better exploit the cost landscape structure, we introduce strategies that enhance local neighborhood exploration and improve optimization success probability. This corresponds to lines 6–23 in Alg. 1.

We achieve this through *linear cost formulations* and *learning rate scheduling*. Initial optimization steps minimize linear formulations of collision costs rather than quadratic alternatives for up to K_{lin} iterations. Linear costs enable more aggressive exploration during initial stages by allowing optimization to persist in high-penalty regions if other variables can achieve lower costs. While quadratic cost growth prevents meaningful navigation of the cost landscape, linear formulations permit escape from local minima, analogous to momentum-based methods. Subsequently, we transition to quadratic cost optimization for K_{quad} steps to achieve fine convergence. As illustrated in Fig. 2, introducing quadratic costs at step 25 causes the loss distributions to diverge distinctly into successful and unsuccessful modes.

The second component for landscape exploration uses learning rate scheduling. The learning rate controls the aggressiveness of exploration away from local minima. High learning rates enable extensive search for strong attractors (likely solution minima) but prevent convergence to these regions. The schedule permits high initial exploration uniformly across state variables, followed by increased convergence near the optimization endpoint. A critical learning rate threshold exists (proportional to the Lipschitz constant and other problem-specific factors) below which the system naturally settles to the nearest local minimum. We minimize time spent in this regime since it provides limited exploration benefit. We provide details on our learning rate schedule and other hyperparameters in Sec. VI.

B. *Trajectory Optimization*

The sampling and optimization procedure from Sec. III-A returns feasible states satisfying placement constraints. In this stage, we use these solutions as initializations and proceed to optimize remaining cost terms, producing feasible robot paths through joint optimization of robot configuration waypoints that form trajectories and object placements.

First, we lift the input object states to joint space representation for each trajectory segment in the motion—this corresponds to a pick or place motion. We accomplish this by converting object poses to joint space coordinates via sampling inverse kinematics on grasp poses. In our work, we use the analytic inverse kinematics for the Franka robot proposed by He and Liu [30], although any other parallel differentiable inverse kinematics is suitable, e.g., [1, 28]. We further initialize our batch of trajectory particles by sampling K_{waypoint} intermediate initialization waypoints between each motion segment's start and goal, taking inspiration from [27] and the retraction state used by cuRobo [1]. We then initialize trajectories by linearly interpolating between the start, intermediate initialization points, and goal for K_{interp} steps for each motion segment. Finally, we execute gradient descent with learning rate scheduling over the entire trajectory using the gradient of the cost functions (specified in Sec. VI) within an augmented Lagrangian optimization framework [31].

Critically, during trajectory optimization, we do not freeze the object placement states derived from the object CSP stage. Instead, placements remain optimizable, and gradients from trajectory-level costs (such as arm-obstacle collisions and kinematic feasibility) propagate through to update object placements. Therefore, if an existing placement is kinematically unreachable or causes arm collisions, the optimizer can shift the placement to a nearby feasible configuration while simultaneously adjusting the trajectory.

IV. Experimental Evaluation

We evaluate SPaSM across a range of sequential manipulation planning problems, including point-to-point motion planning tasks, sequential pick-and-place tasks for packing and tower stacking in clutter, and real-time adaptation scenarios. These problems vary significantly in difficulty, with particular emphasis on tasks featuring tightly coupled constraints and high nonlinearity. We compare our approach against state-of-the-art baselines cuTAMP [2] and cuRobo [1] (without jerk minimization) across various batch sizes. All experiments were conducted on an x86-based desktop computer with an AMD Ryzen Threadripper PRO 5965WX 24-Core CPU and an NVIDIA GeForce RTX 4090 GPU.

A. *Point-to-Point Motion Planning Problems*

The general formulation of SPaSM enables its application to classical motion planning problems by treating them as single-motion trajectory optimization instances. These problems can be conceptualized as single-sequence-length manipulation tasks, with constraints imposed on feasible configurations by environment geometry. We evaluate SPaSM with its trajectory optimization component (SPaSM + TrajOpt) on

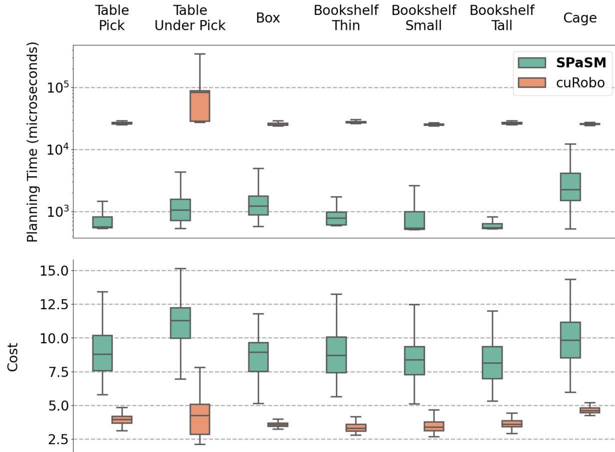


Fig. 3: Comparison of SPaSM with cuRobo on the MotionBenchMaker dataset. The success rate of both planners is 100%. SPaSM is more than an order of magnitude faster, but as expected, found solution paths are longer.

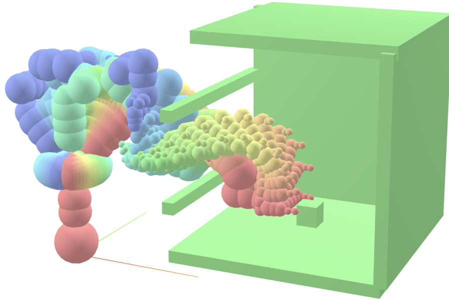


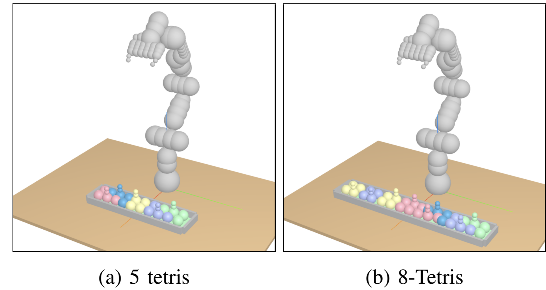
Fig. 4: Cutaway with an example trajectory generated for an instance of the "cage" problem from MotionBenchMaker [32]. SPaSM is able to quickly generate collision free trajectories in complex environments.

the MotionBenchMaker dataset [32] for the 7-DoF Franka Panda. We compare against cuRobo [1], a state-of-the-art GPU-accelerated trajectory optimization-based motion planner, measuring both runtime and solution path length.

Fig. 3 presents the comparative results across different problem instances. SPaSM demonstrates superior planning time performance, achieving solution times more than an order of magnitude faster than cuRobo while maintaining the same 100% success rate as cuRobo. As expected, the solution paths generated by SPaSM are longer than those produced by cuRobo, reflecting a trade-off between optimization speed and path optimality. These results demonstrate SPaSM's capability to solve difficult planning problems within a unified framework. Motion planning and manipulation planning can be solved by the same method without losing performance.

B. Pick-and-Place: Tetris Packing

We evaluate SPaSM on a sequential pick-and-place task requiring precise object placement configurations, based on the 5-Tetris benchmark introduced by cuTAMP [2], shown in Fig. 5. We evaluate scenarios with 5 and 8 Tetris blocks: the planner must determine feasible placements for the blocks that fit within a tight rectangular bounding box (Fig. 5). The blocks are designed such that satisfying packings utilize all available space without gaps, emphasizing



(a) 5 tetris (b) 8-Tetris

Fig. 5: 5-Tetris and 8-Tetris problems. The goal is to pack the blocks into the bounding box. This problem requires long horizon planning, as initial placements greatly the availability of a solution and placement positions of later blocks.

Approach	N_b	Success (%)	Sol. Time
SPaSM w/o TrajOpt (tuned)	512	100	4.23 ± 2.68 ms
	1024	100	**4.12 ± 1.94 ms**
	2048	100	4.56 ± 1.71 ms
	4096	100	6.37 ± 1.97 ms
	8192	100	12.66 ± 2.11 ms
cuTAMP only (tuned)	512	100	18.98 ± 6.80 s
	1024	100	12.14 ± 2.92 s
	2048	100	**7.33 ± 0.93 s**
	4096	100	9.01 ± 1.10 s
	8192	100	11.88 ± 0.98 s

(a) 5-Tetris

Approach	N_b	Success (%)	Sol. Time
SPaSM w/o TrajOpt (Tuned)	512	100	**12.68 ± 7.40 ms**
	1024	100	14.15 ± 6.03 ms
	2048	100	17.45 ± 5.10 ms
	4096	100	36.71 ± 3.70 ms
	8192	100	88.97 ± 3.77 ms
cuTAMP only (Tuned)	512	80.0	**65.48 ± 30.38 s**
	1024	83.33	133.69 ± 61.31 s
	2048	100	119.91 ± 49.12 s
	4096	100	74.58 ± 15.15 s
	8192	100	90.06 ± 19.74 s

(b) 8-Tetris

TABLE I: Comparison for the 5-Tetris and the 8-Tetris problems in terms of success (coverage) and solution time. Note that for SPaSM times are reported in **milliseconds** and for cuTAMP in seconds.

that valid solutions constitute effectively a sparsely distributed zero-dimensional subset in the configuration space. To demonstrate scalability, we developed an 8-block variant extending the original benchmark. For SPaSM, the cost function combines inter-object penetration penalties with boundedness constraints within the specified region.

1) Baseline Configuration: Since cuTAMP does not perform trajectory optimization within its standard pipeline, we combine it with cuRobo as recommended by the original authors to generate complete motion plans. We present cuTAMP results with cuRobo trajectory optimization. To ensure a fair comparison, we initialize cuTAMP with a predetermined skeleton known to yield feasible solutions, and do not count the time for this initialization. cuTAMP was configured with tuned cost parameters and all visualization disabled for performance. For results incorporating

Problem	Approach	Sol. Time	Sol. Length
5-TETRIS	SPaSM + TrajOpt	**10.03 ± 0.95 ms**	**17.25 ± 0.36**
	cuTAMP + cuRobo	10.55 ± 1.90 s	28.24 ± 2.47
8-TETRIS	SPaSM + TrajOpt	**19.39 ± 2.06 ms**	**26.03 ± 0.53**
	cuTAMP + cuRobo	69.42 ± 32.12 s	48.75 ± 3.15

TABLE II: Comparison for the full 5-Tetris problem with joint optimization. Both methods have a 100% success rate. We choose the best batch size based on the results from Fig. 6.

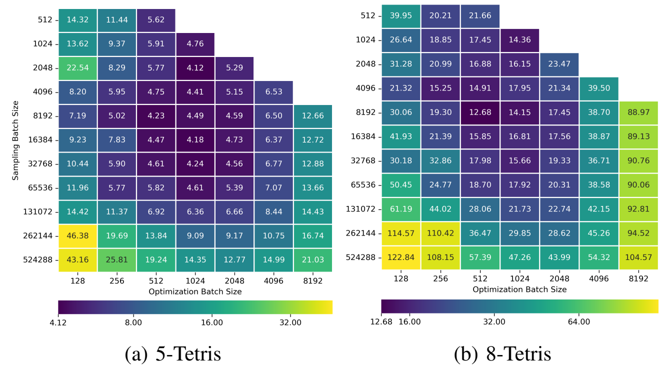
trajectory optimization, we report runtime and solution cost metrics only for the best-performing batch from stage one, as the batch size for trajectory optimization remains constant regardless of sampling and optimization batch configurations.

2) Performance Comparison: For each problem instance, we conducted 100 trials per method and report the time to first solution with 95% confidence intervals. While continued optimization would yield additional solutions, we focus on the more stringent metric of time to first solution, with a maximum limit of 30,000 optimization steps. Against the cuTAMP and cuRobo baseline, we present results for SPaSM with trajectory optimization and SPaSM without trajectory optimization (object CSP stage only). For SPaSM without trajectory optimization, success is measured solely on placement feasibility, excluding trajectory considerations.

Tables Ia and Ib demonstrate that SPaSM successfully finds solutions in all test cases, while cuTAMP fails on some instances of the 8-block tetris problem. Altogether, SPaSM with trajectory optimization achieves solution times more than 1,000 times faster than cuTAMP combined with cuRobo(Table II). More significantly, SPaSM without trajectory optimization finds feasible placements nearly **4,000 times faster** than cuTAMP alone, demonstrating SPaSM's exceptional efficiency in computing feasible object placements. In addition, SPaSM generates shorter trajectories due to the joint optimization setup.

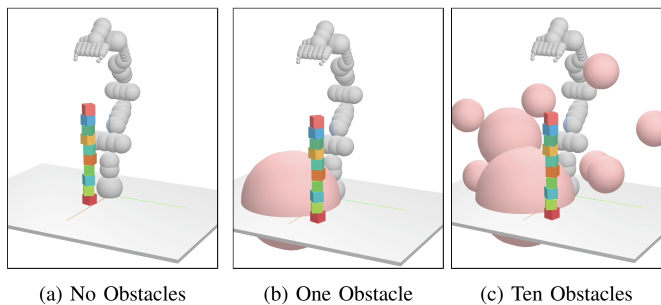
3) Hyperparameter Ablations: We conducted ablation studies to determine optimal sampling batch size N and optimization batch size M for SPaSM without trajectory optimization. We performed hyperparameter search over sampling batch sizes $\in 2^{[9,19]}$ and optimization batch sizes $\in 2^{[8,13]}$. Each parameter combination was evaluated across 100 trials with average solution times recorded. Our efficient runtime enables such hyperparameter exploration.

The analysis revealed unexpected relationships between these parameters (Figs. 6a and 6b). Sampling incurs significantly lower computational cost than optimization (1 cost evaluation per batch versus 30 gradient evaluations per batch). However, since only a single solution is required, excessive sampling wastes computational resources by generating redundant solutions. The search identified two distinct optimization strategies for minimizing time to first solution: (1) small batch sizes for both parameters, enabling fast iteration with low re-solving penalties, and (2) large batch sizes for both parameters, providing high success probability on initial attempts. The more challenging 8-Tetris problem performed better with larger batch sizes, likely correlating with the increased sparsity of solution states.



(a) 5-Tetris (b) 8-Tetris

Fig. 6: Hyperparameter sweep over sampling batch and optimization batch sizes (N and M) and average solution times for 5-Tetris and 8-Tetris. There is a balance between high batch sizes (which have higher chance of solution for one solve step, but are computationally heavy) and smaller batch sizes (which have less likelyhood of solution but take less time).



(a) No Obstacles (b) One Obstacle (c) Ten Obstacles

Fig. 7: Tower stacking problem. The goal is to stack blocks on the table into a tower while avoiding obstacles. Solving this problem requires joint optimization of placement and trajectory to find feasible end-to-end motions.

C. Pick-and-Place: Tower Stacking

Combining the computational challenges from point-to-point motion planning and discrete placement optimization, we designed a tower stacking problem involving long-horizon sequential manipulation under environmental constraints. We address the task of stacking $B = 10$ blocks into a physically stable tower configuration. We evaluate three problem variants: without obstacles, with a single obstacle, and with multiple obstacles (Fig. 7). The obstacles constrain the robot's motion, rendering many potential tower placements infeasible. Our joint optimization framework simultaneously solves for stable block placement (incorporating center-of-mass constraints and collision-free object

Num. Obs.	Sol. Time	Sol. Length	Particle Opt. Time	TrajOpt Time
0	10.80 ± 0.07	33.21 ± 0.84	0.39 ± 0.01	10.42 ± 0.06
1	10.75 ± 0.07	32.49 ± 0.66	0.38 ± 0.01	10.37 ± 0.06
10	12.76 ± 0.06	32.99 ± 0.70	0.40 ± 0.01	12.36 ± 0.05

TABLE III: Ablation studies for the 10-block tower stacking problem with different number of obstacles (Fig. 7). It is interesting to note that SPaSM+TrajOpt scales well even with increasing environment complexity.

positioning) and collision-free arm trajectories, thus enabling solutions in cluttered environments.

For the tower stacking problem, even without obstacles, SPaSM without trajectory optimization consistently fails because interpolated straight-line trajectories inevitably collide with previously placed blocks. This constraint emphasizes the necessity of trajectory optimization for sequential manipulation tasks. Following the evaluation done for tetris, we benchmark runtimes and solution lengths across all three problem variants (Table III).

Our method successfully solves the multiple-obstacle configuration with a 100% success rate, demonstrating robustness to environmental clutter. Table III shows that solution times remain consistent across obstacle configurations, indicating that our approach scales effectively with increasing complexity. The particle optimization phase requires approximately 0.39 milliseconds regardless of obstacle density, while trajectory optimization time takes between 10 and 13 milliseconds. This consistency demonstrates SPaSM's scalability and robustness to complex environments and tasks.

V. Conclusion and Future Work

We present Sampling Particle optimization for Sequential Manipulation (SPaSM), a fully GPU-parallelized method for sequential pick-and-place planning tasks in clutter that achieves solution times on the order of *milliseconds* on challenging benchmark problems—a $4000\times$ speedup compared to existing approaches [2]. Our end-to-end compilation of constraint evaluation, sampling, and gradient-based optimization into optimized CUDA kernels eliminates CPU-GPU coordination overhead that previously limited accelerated planning approaches. Our approach also jointly optimizes object placements and robot trajectories in a two-stage approach, demonstrating that the coupling between placements and motion requires integrated optimization to solve in cluttered environments.

The millisecond planning horizon achieved by SPaSM on challenging benchmarks questions assumptions about what could be planned online versus offline—SPaSM transforms sequential manipulation into a real-time planning capability, enabling systems to respond to changes in the environment with replanning frequencies sufficient for dynamic adaptation. This will enable entirely new applications of complex manipulation, including human-robot collaboration, reactive behaviors, and closed-loop control informed by long-horizon information. Future research directions include integration with perception systems for reactive planning in the real world, extension to receding horizon formulations that integrate robot dynamics into the problem formulation, and incorporation of learned components to avoid collisions and draw potential placement and grasp samples. We also wish to investigate a multi-robot sequential planning (e.g., as in Lai et al. [33]) extension of SPaSM.

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, et al. "Curobo: Parallelized collision-free robot motion generation". In: *IEEE Int. Conf. Robot. Autom.* 2023, pp. 8112–8119.

[2] W. Shen, C. R. Garrett, N. Kumar, A. Goyal, T. Hermans, L. P. Kaelbling, T. Lozano-Pérez, and F. Ramos. "Differentiable GPU-Parallelized Task and Motion Planning". In: *Robotics: Science and Systems*. LosAngeles, CA, USA, June 2025.

[3] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.

[4] M. Görner, R. Haschke, H. Ritter, and J. Zhang. "Moveit! task constructor for task-level motion planning". In: *Int. Conf. Robot. Autom.* 2019, pp. 190–196.

[5] P. Englert, I. M. Rayas Fernández, R. K. Ramachandran, and G. Sukhatme. "Sampling-Based Motion Planning on Sequenced Manifolds". In: *Robotics: Science and Systems*. Virtual, July 2021.

[6] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. "Integrated task and motion planning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1 (2021), pp. 265–293.

[7] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning". In: *Int. Conf. on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 440–448.

[8] C. Quintero-Pena, Z. Kingston, T. Pan, R. Shome, A. Kyrillidis, and L. E. Kavraki. "Optimal grasps and placements for task and motion planning in clutter". In: *IEEE Int. Conf. Robot. Autom.* IEEE, 2023.

[9] 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". In: *The Int. Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[10] M. Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." In: *IJCAI*. 2015, pp. 1930–1936.

[11] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018.

[12] C. V. Braun, J. Ortiz-Haro, M. Toussaint, and O. S. Oguz. "Rhh-lgp: Receding horizon and heuristics-based logic-geometric programming for task and motion planning". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2022, pp. 13761–13768.

[13] M. Toussaint, J. Harris, J.-S. Ha, D. Driess, and W. Hönig. "Sequence-of-constraints MPC: Reactive timing-optimal control of sequential manipulation". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2022, pp. 13753–13760.

[14] L. Yan, T. Stouraitis, J. Moura, W. Xu, M. Gienger, and S. Vijayakumar. "Impact-aware bimanual catching of large-momentum objects". In: *IEEE Transactions on Robotics* 40 (2024), pp. 2543–2563.

[15] Y. Zhang, C. Pezzato, E. Trevisan, C. Salmi, C. H. Corbato, and J. Alonso-Mora. "Multi-modal mppi and active inference for reactive task and motion planning". In: *IEEE Robot. Autom. Lett.* (2024).

[16] T. Lozano-Pérez and P. A. O'Donnell. "Parallel robot motion planning." In: *IEEE Int. Conf. Robot. Autom.* 1991, pp. 1000–1007.

[17] W. Thomason, Z. Kingston, and L. E. Kavraki. "Motions in microseconds via vectorized sampling-based planning". In: *IEEE Int. Conf. Robot. Autom.* 2024, pp. 8749–8756.

[18] N. M. Amato and L. K. Dale. "Probabilistic roadmap methods are embarrassingly parallel". In: *IEEE Int. Conf. Robot. Autom.* Vol. 1. 1999, pp. 688–694.

[19] J. Bialkowski, S. Karaman, and E. Frazzoli. "Massively parallelizing the RRT and the RRT". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2011, pp. 3513–3518.

[20] J. Pan and D. Manocha. "GPU-based parallel collision detection for fast motion planning". In: *The Int. Journal of Robotics Research* 31.2 (2012), pp. 187–200.

[21] C. H. Huang, P. Jadhav, B. Plancher, and Z. Kingston. "prrtc: Gpu-parallel rrt-connect for fast, consistent, and low-cost motion planning". In: *arXiv preprint 2503.06757* (2025).

[22] Q. Wu, F. Xiong, F. Wang, and Y. Xiong. "Parallel particle swarm optimization on a graphics processing unit with application to

trajectory optimization". In: *Engineering Optimization* 48.10 (2016), pp. 1679–1692.
[23] Z. Pan, B. Ren, and D. Manocha. "Gpu-based contact-aware trajectory optimization using a smooth force model". In: *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2019, pp. 1–12.
[24] S. Heinrich, A. Zoufahl, and R. Rojas. "Real-time trajectory optimization under motion uncertainty using a GPU". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2015, pp. 3572–3577.
[25] X. Bu and B. Plancher. "Symmetric stair preconditioning of linear systems for parallel trajectory optimization". In: *IEEE Int. Conf. Robot. Autom.* 2024, pp. 9779–9786.
[26] A. T. Le, G. Chalvatzaki, A. Biess, and J. R. Peters. "Accelerating motion planning via optimal transport". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 78453–78482.
[27] A. T. Le, K. Hansel, J. Carvalho, J. Watson, J. Urain, A. Biess, G. Chalvatzaki, and J. Peters. "Global tensor motion planning". In: *IEEE Robot. Autom. Lett.* (2025).
[28] C. M. Kim, B. Yi, H. Choi, Y. Ma, K. Goldberg, and A. Kanazawa. "PyRoki: A Modular Toolkit for Robot Kinematic Optimization". In: *arXiv preprint 2505.03728* (2025).
[29] Y. Lee, A. Z. Li, P. Huang, E. Heiden, K. M. Jatavallabhula, F. Damken, K. Smith, D. Nowrouzezahrai, F. Ramos, and F. Shkurti. "STAMP: Differentiable Task and Motion Planning via Stein Variational Gradient Descent". In: *IEEE Robot. Autom. Lett.* 10.6 (2025), pp. 6007–6014.
[30] Y. He and S. Liu. "Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design". In: *Int. Conf. on Control, Mechatronics Autom.* 2021, pp. 194–199.
[31] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 2006.
[32] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. "Motionbenchmaker: A tool to generate and benchmark motion planning datasets". In: *IEEE Robot. Autom. Lett.* 7.2 (2021), pp. 882–889.
[33] M. Lai, K. Go, Z. Li, T. Kröger, S. Schaal, K. Allen, and J. Scholz. "RoboBallet: Planning for multirobot reaching with graph neural networks and reinforcement learning". In: *Science Robotics* 10.106 (2025), eads1204.

VI. Appendix

A. Tetris Problem

The placement cost function is defined as:

$$C_{\text{placement}} = \sum_{i \in B} \sum_{j > i \in B} P_{(i,j)} + \sum_{i \in B} \sum_{j \in W} P_{(i,j)} + |z - z^*| \quad (1)$$

where $P_{(i,j)}$ represents the penetration depth between objects i and j. The cost includes penetration penalties between block pairs, block-wall pairs, and the distance from the desired z-axis position. B denotes the set of blocks, W denotes the set of walls, and z^* is the target z-coordinate.

B. Tower Stacking Problem

The initial placement of blocks is determined by minimizing a cost function $C_{\text{placement}}$ that evaluates the quality of a tower configuration $\{\mathbf{p}_i\}_{i \in B}$. Here, $\mathbf{p}_i$ represents the pose of block i. The cost function is a weighted sum of penalties for instability, incorrect block height, and collisions:

$$C_{\text{placement}} = w_{\text{stable}} C_{\text{stable}} + w_{\text{height}} C_{\text{height}} + w_{\text{coll}} C_{\text{coll}} \quad (2)$$

Stability Cost (C_{stable}): Penalizes configurations where the center of mass (CoM) of blocks stacked above block i projects outside its supporting footprint.

$$C_{\text{stable}} = \sum_{i=1}^{B-1} \text{Dist}(\text{CoM}(\{\mathbf{p}_j\}_{j>i}), \text{Footprint}(\mathbf{p}_i)) \quad (3)$$

Height Cost (C_{height}): Enforces that each block i is placed at a height corresponding to its index in the stack.

$$C_{\text{height}} = \sum_{i=1}^{B} (z(\mathbf{p}_i) - i \cdot h_{\text{block}})^2 \quad (4)$$

Collision Cost (C_{coll}): Penalizes penetration between block pairs and between blocks and environmental obstacles $\mathcal{O}$.

$$C_{\text{coll}} = \sum_{i,j>i \in B} \text{Pen}(\mathcal{B}_i, \mathcal{B}_j) + \sum_{i \in B, o \in \mathcal{O}} \text{Pen}(\mathcal{B}_i, o) \quad (5)$$

In these equations, $z(\mathbf{p}_i)$ is the vertical position of block i, h_{block} is the height of a single block, and $\text{Pen}(\cdot,\cdot)$ is a penetration depth function. The optimization is performed over sequences of joint configurations $\mathbf{q}_{b,t} \in \mathbb{R}^7$ for each of the B blocks at each timestep $t \in \{0, \ldots, T\}$:

$$C_{\text{trajopt}} = C_{\text{placement}}(\{\mathbf{p}_{b,T}\}_{\forall b}) + w_{\text{start}} C_{\text{start}} + w_{\text{arm}} C_{\text{arm}} + w_{\text{block}} C_{\text{block}}$$

Placement Cost ($C_{\text{placement}}$): Evaluates the stability of the final tower configuration. Final block poses $\{\mathbf{p}_{b,T}\}_{\forall b}$ are determined by forward kinematics at the final timestep: $\text{fk}(\mathbf{q}_{b,T})$.

Initial Pose Alignment Cost (C_{start}): Penalizes misalignment between the arm's end-effector and the block's initial pose at $t = 0$ to ensure smooth grasping. The cost measures squared error in position and roll/pitch orientation:

$$C_{\text{start}} = \sum_{b=1}^{B} \|\text{pos}(\text{fk}(\mathbf{q}_{b,0})) - \text{pos}(\mathbf{p}_{b,\text{initial}})\|_2^2 + \quad (6)$$

$$\|\text{rot}_{\text{RP}}(\text{fk}(\mathbf{q}_{b,0})) - \text{rot}_{\text{RP}}(\mathbf{p}_{b,\text{initial}})\|_2^2 \quad (7)$$

Arm-Obstacle Collision Cost (C_{arm}): Penalizes penetration depth between robot arm geometry $\mathcal{A}(\mathbf{q}_{b,t})$ and workspace obstacles $o \in \mathcal{O}$ across the entire trajectory:

$$C_{\text{arm}} = \sum_{b=1}^{B} \sum_{t=0}^{T} \sum_{o \in \mathcal{O}} \text{Pen}(\mathcal{A}(\mathbf{q}_{b,t}), o) \quad (8)$$

Held Block-Obstacle Collision Cost (C_{block}): Penalizes penetration depth between held block geometry $\mathcal{G}(\text{fk}(\mathbf{q}_{b,t}))$ and obstacles $o \in \mathcal{O}$ during transport phase ($t \in [1, T-1]$):

$$C_{\text{block}} = \sum_{b=1}^{B} \sum_{t=1}^{T-1} \sum_{o \in \mathcal{O}} \text{Pen}(\mathcal{G}(\text{fk}(\mathbf{q}_{b,t})), o) \quad (9)$$

The scalar weights w_{start}, w_{arm}, and w_{block} balance the contribution of each term. Minimizing C_{trajopt} yields joint-space trajectories that achieve stable tower configurations while respecting physical and environmental constraints.