MASKED GENERATIVE MOTION PLANNING WITH GEOMETRY-GUIDED TOKEN SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative motion planners typically use learned trajectory priors for initial generation, while leaving test-time repair to local continuous refinement. We introduce Masked Generative Motion Planning (MGMP), which extends the learned prior from efficient parallel generation to structural repair. A masked generative transformer generates discrete trajectory candidates in parallel, and *Geometry-Guided Token Search (GGTS)* uses scene geometry to target where to edit and which prior-supported alternatives to evaluate. This turns refinement into an efficient search over discrete motion alternatives, enabling route-level restructuring beyond local trajectory deformation. MGMP achieves 96% success on *Ring Maze* and 82% repair success on *Controlled Route Invalidation on Kuka*, exceeding the strongest external baselines by 23 and 25 percentage points, respectively. It further generalizes to unseen layouts, additional obstacles, unseen geometries, single- and dual-arm planning, and real-world Baxter tasks. Further results and videos are available at: `https://anonymous.4open.science/r/MGMP_GGTS-5FC6/`.

1 INTRODUCTION

Motion planning in robotics aims to find a feasible trajectory between a start and goal configuration while satisfying geometric constraints (LaValle, 2006). Trajectory-optimization methods directly refine trajectories against task-specific objectives (Ratliff et al., 2009; Schulman et al., 2014), but their performance is typically sensitive to initialization in cluttered and non-convex environments (Pan et al., 2014). Generative planners mitigate this sensitivity by learning multimodal trajectory priors and producing diverse candidate motions that can subsequently be refined to satisfy collision constraints. Generative motion planning therefore involves two coupled inference problems: *efficient initial trajectory generation* and *effective test-time geometric refinement*.

Existing planners make trade-offs on both fronts. For *initial generation*, diffusion-based methods rely on iterative denoising in a high-dimensional continuous space (Janner et al., 2022; Carvalho et al., 2023), leading to high inference cost; fewer denoising steps improve efficiency but can reduce sample quality. For *test-time refinement*, generated trajectories are typically refined through continuous guidance or trajectory optimization (Carvalho et al., 2023; Luo et al., 2024; Yang et al., 2025). Weak guidance may leave collisions unresolved, while stronger guidance can move trajectories away from the learned prior (Yang et al., 2025). Such continuous refinement is effective when feasibility can be recovered through local adjustments; however, in harder cases such as near-enclosing obstacles, reaching a feasible alternative may require many iterations, remain sensitive to initialization and guidance strength, and still fail to escape unfavorable local basins. (Pan et al., 2014; Luo et al., 2024).

We therefore ask: *How can a learned trajectory prior directly expose qualitatively different routes as candidates for structural repair, rather than expecting continuous refinement to reach them?*

We introduce Masked Generative Motion Planning (MGMP), which addresses this question by formulating motion planning as masked generation over discrete trajectory tokens, reusing the same token space for test-time structural repair. A trajectory tokenizer learns compact motion tokens from feasible trajectories. For initial generation, a Masked Generative Transformer (MGT) (Chang et al., 2022; 2023) predicts token sequences in parallel, producing high-quality trajectory candidates in a single forward pass and reducing the need for costly resampling and post-hoc repair during online planning. Importantly, the same representation allows us to cast repair as a guided search over prior-supported motion alternatives, instead of continuously deforming the current trajectory.

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

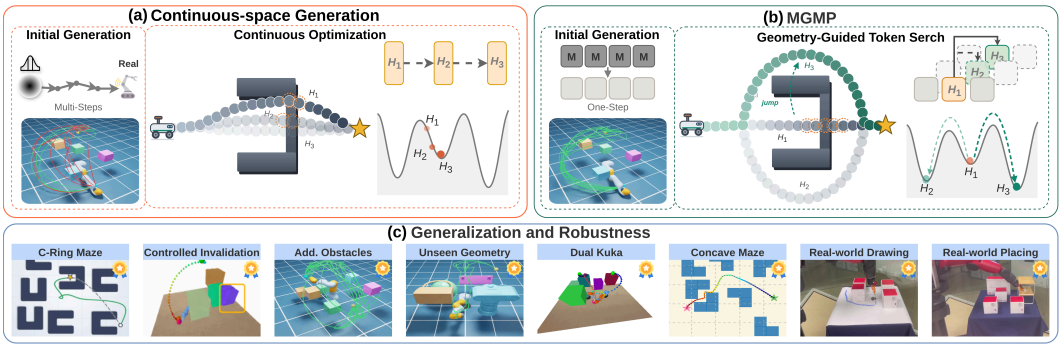


Figure 1: Overview of *MGMP*. We replace continuous trajectory generation and refinement with discrete masked generation and Geometry-Guided Token Search, enabling efficient planning and structural route repair. We evaluate it across challenging simulated and real-world settings, including basin escape, unseen geometries, additional obstacles, dual-arm planning, and Baxter tasks.

To make this search effective, we introduce **Geometry-Guided Token Search (GGTS)**, which uses scene geometry to guide two decisions: *where to edit* and *what to fill*. Geometric attribution determines *where to edit* by selecting the token positions most responsible for the current violation while preserving unaffected parts of the trajectory. To decide *what to fill*, the MGT proposes alternative motion tokens within the selected repair region, which are first screened by a geometric score and then verified through exact trajectory evaluation before an improving edit is finalised. This targeted, coarse-to-exact search avoids unguided perturbations and exhaustive codebook scans; instead, a small number of targeted discrete replacements induce large, coordinated changes in motion structure, enabling exploration of alternative routes beyond the current planning basin.

We validate MGMP on diverse tasks and environments, showing that it generalizes across unseen layouts, additional obstacles and unseen geometries, scales to 14-DoF dual-arm planning, and transfers to real-world Baxter tasks. We also introduce two diagnostic benchmarks: *Ring Maze*, which requires substantial route changes to escape local planning basins, and *Controlled Route Invalidation*, which fixes the same invalidated trajectory across methods to isolate repair from initial-generation quality. MGMP achieves 96% success on Ring Maze and 82% repair success on Controlled Route Invalidation, exceeding the strongest external baselines by 23 and 25 percentage points, respectively.

In summary, this paper makes the following three contributions:

- We introduce MGMP, which reframes generative motion planning by extending the learned trajectory prior beyond efficient parallel generation to test-time structural repair.
- We propose *Geometry-Guided Token Search (GGTS)*, which turns test-time refinement into an efficient, targeted search over discrete motion alternatives, using scene geometry to determine where to edit and which candidates to evaluate, enabling structural repair when local continuous refinement is insufficient.
- We introduce *Ring Maze* and *Controlled Route Invalidation* to evaluate basin escape and isolate structural repair from initial-generation quality, and validate MGMP across unseen obstacle layouts, additional obstacles, unseen geometries, and real-world Baxter tasks.

2 Related Work

Classical and learning-based motion planning. Classical planners either explore configuration space or locally refine a trajectory. Sampling-based methods (Kavraki et al., 1996; LaValle, 1998; Karaman & Frazzoli, 2011; Gammell et al., 2015) can discover alternative routes but often require extensive sampling and collision checking in cluttered or high-dimensional spaces (Orthey et al., 2024; Kavraki et al., 1996; Hauser, 2015). Trajectory optimizers (Ratliff et al., 2009; Zucker et al., 2013; Kalakrishnan et al., 2011; Schulman et al., 2013; Mukadam et al., 2016) efficiently refine collision and smoothness objectives, but remain sensitive to initialization and local planning basins (Pan et al., 2014; Schulman et al., 2014). This motivates learning reusable structure across planning problems (Qureshi et al., 2021; Fishman et al., 2023; Das et al., 2024). Learning-based planners reduce online planning cost by learning sampling distributions, predictors, policies, or latent

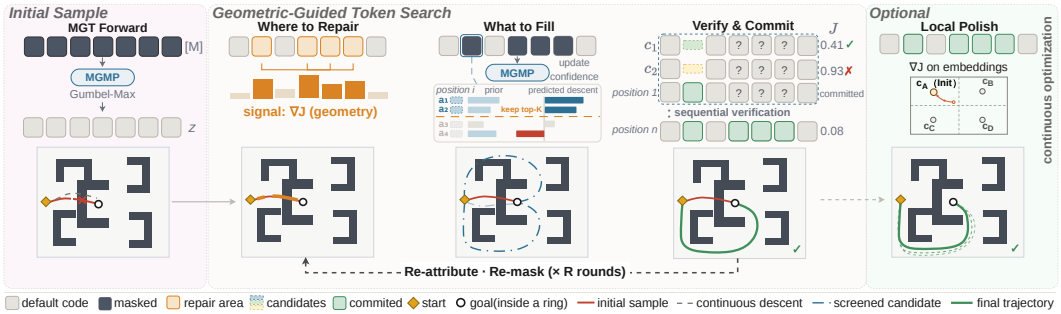


Figure 2: Overview of GGTS. We use geometric attribution to select edit positions, first-order scores to screen token replacements, and sequential verification to accept improving candidates, followed by optional continuous polish.

representations (Qureshi et al., 2021; Qureshi & Yip, 2018; Yu & Gao, 2021; Fishman et al., 2023; Yamada et al., 2023). Generative planners further model multimodal trajectory distributions, with diffusion-based methods producing diverse candidates through iterative denoising (Janner et al., 2022; Carvalho et al., 2023; Luo et al., 2024). Because generated trajectories may violate scene constraints, they commonly require guidance, correction, or trajectory optimization (Carvalho et al., 2023; Luo et al., 2024; Seo et al., 2025; Yang et al., 2025). Such continuous refinement is effective for local correction but typically cannot move from an infeasible region of configuration space to a feasible region when this requires a substantial route change to escape a local optimum.

Discrete representations and masked generative models. Discrete latent representations have been used to compress actions and trajectories for planning and learned sampling (Jiang et al., 2023; Johnson et al., 2023). Masked generative models instead predict multiple discrete tokens in parallel and iteratively revisit selected tokens over only a few refinement steps (Chang et al., 2022; 2023). This paradigm has been applied to trajectory modeling, model-predictive control, human motion generation, and robotic imitation learning (Wu et al., 2023; Wen et al., 2025; Pinyoanuntapong et al., 2024; Guo et al., 2024; Zhuang et al., 2025). We extend masked generation to collision-constrained robot motion planning, using learned motion tokens for efficient parallel trajectory generation.

Test-time trajectory refinement and repair. Generated trajectories are commonly adapted to scene-specific constraints through gradient-based guidance or trajectory optimization (Ratliff et al., 2009; Kalakrishnan et al., 2011; Schulman et al., 2014; Carvalho et al., 2023; Luo et al., 2024; Yang et al., 2025). These methods are effective for local geometric correction, but remain sensitive to initialization when feasibility requires a global route change (Pan et al., 2014; Schulman et al., 2014). Recent masked generative methods revisit selected tokens during inference, but their update decisions are designed for generic generative quality rather than scene-specific geometric constraint violations (Peng et al., 2025; Liu et al., 2025). Beyer and Karaman Beyer & Karaman (2026) instead search compressed trajectory tokens under test-time objectives, but use fixed coarse-to-fine traversal and candidate evaluation rather than a learned conditional prior to focus token proposals or geometric failure signals to target repair. MGMP instead turns the learned trajectory prior itself into a geometry-guided, targeted search space for efficient structural repair over prior-supported route alternatives.

3 Methodology

We formulate motion planning as conditional trajectory generation. Given a start state $c_s \in \mathbb{R}^d$, a goal state $c_g \in \mathbb{R}^d$, and environment constraints w, the planner is required to generate feasible motion trajectories $\{x_t\}_{t=0}^{T-1}$. Here d is the dimensionality of the robot's state space, x_t is its configuration at frame t, and T is the horizon of the full trajectory. Sampled trajectories must satisfy $x_0 = c_s$ and $x_{T-1} = c_g$, and the entire trajectory must be collision-free with respect to the environment.

Our method MGMP consists of a trajectory tokenizer and a conditional masked generative transformer (MGT). The tokenizer maps expert trajectories into discrete codes and reconstructs continuous trajectories (Sec. 3.1), while the MGT predicts masked codes conditioned on the start state, goal state, and environment (Sec. 3.2). At test time, Geometry-Guided Token Search (GGTS) uses the learned predictor together with geometric feedback to repair the token trajectory (Fig. 2; Sec. 3.3).

3.1 Representing Motion Configuration Space via Codebooks

To construct a compact discrete representation of feasible trajectories, we use a VQ-VAE (Van Den Oord et al., 2017) with a learnable codebook $\mathcal{K}$. The encoder maps $x \in \mathbb{R}^{T\times d}$ to latent representations $\hat{\mathbf{e}} \in \mathbb{R}^{N\times d'}$, which are quantized by nearest-neighbor lookup to discrete token indices $\mathbf{z} = (z_1, \ldots, z_N)$. The codebook is $\mathcal{K} = \{\mathbf{k}_m\}_{m=0}^{|\mathcal{K}|-1}$, $\mathbf{k}_m \in \mathbb{R}^{d'}$, with quantized embedding $\mathbf{e}_i = \mathbf{k}_{z_i}$ at position i. Each token summarizes approximately T/N trajectory steps, and a symmetric decoder reconstructs the continuous trajectory $\hat{x}$ from the quantized embeddings $\mathbf{e}$. Our VQ-VAE is trained with the reconstruction and commitment objective (Van Den Oord et al., 2017), $\mathcal{L}_{\text{VQ}} = \lambda_{\text{rec}}\|x - \hat{x}\|_1 + \beta\|\hat{\mathbf{e}} - \text{sg}[\mathbf{e}]\|_2^2$, together with a velocity loss to encourage temporal continuity and, for robot trajectories, a forward-kinematics loss. After training, the tokenizer is frozen: the encoder maps expert trajectories to discrete tokens for MGT training, while at inference time predicted tokens are mapped to codebook embeddings and decoded into continuous trajectories (See App. B.2).

3.2 Masked Generative Predictor over Codebooks

We use a masked generative transformer (MGT) to model trajectories in the discrete space learned by the tokenizer. The MGT is an encoder-only transformer (Vaswani et al., 2017) that predicts masked trajectory tokens conditioned on the start state c_s, goal state c_g, and obstacle information w. Start and goal states are projected into condition tokens, while obstacle centers are encoded by a lightweight transformer and injected into the MGT through AdaLN conditioning. During training, trajectories are first encoded into discrete tokens. A random subset is replaced by mask tokens, while some remaining tokens are randomly corrupted to alternative codebook indices. The MGT is trained with cross-entropy loss to recover the original token indices at the masked or corrupted positions, conditioned on the start, goal, and obstacle information. Details are provided in App. B.2.

3.3 Geometry-Guided Token Search

The trained mask transformer (Sec. 3.2) provides a strong prior over trajectory tokens. Given the planning condition $\mathbf{c} = (c_s, c_g, w)$, we obtain an initial trajectory through a single forward pass of the MGT, based on probability confidence; however, a high-confidence token can still produce a collision after decoding. We therefore formulate test-time refinement as a *constraint-guided search in the learned discrete trajectory space*. Following the plan-and-denoise view of masked generation (Peng et al., 2025; Liu et al., 2025), each refinement round makes two decisions: which positions to re-mask (Sec. 3.3.1) and what codes to put there (Sec. 3.3.2). Rather than relying solely on predictive uncertainty, we use an explicit geometric objective J to guide both decisions, while the MGT prior preserves learned motion structure and smoothness. At each editable position, the conditional MGT restricts the search to motion codes with high probability under the learned trajectory distribution, while geometry determines which positions and replacements are worth evaluating. This retains the ability to make larger discrete changes in trajectory structure while explicitly accounting for geometric constraints. The task-specific form of J is detailed in App. A.2.

3.3.1 Where to Edit: Geometry-Guided Adaptive Remasking

The first MGT forward pass defines the token distribution $p_\theta(z_i \mid \mathbf{c})$ at each position. We initialize each token z_i using Gumbel-Max sampling. We employ the *Gauss-Southwell rule* to select the regions requiring refinement—choosing those tokens where the magnitude of the gradient is greatest. Specifically, we backpropagate J through the decoder to the current token embeddings $\mathbf{e}_i$ and define

$$s_i = \left\| \frac{\partial J}{\partial \mathbf{e}_i} \right\|_2 \left(1 + \lambda_{\text{conf}}(1 - c_i)\right), \tag{1}$$

where c_i is the MGT confidence of token i and λ_{conf} is a scale factor. The geometric gradient provides the primary signal, while confidence provides a secondary modulation. Thus, a high-confidence token can still be replaced when it strongly contributes to a geometric violation.

We rank the token positions by s_i and re-mask the smallest top-ranked set whose cumulative score accounts for a fraction ρ of the total score. Token positions associated with detected collision timesteps are additionally included in the re-masking set. The mask size thus adapts to the failure: a shallow contact considers replacing only a few tokens, while a trajectory on the wrong side of an

obstacle exposes a larger set of trajectory coordinates, which enables substantial restructuring. Such changes can enable substantial route changes (see Sec. 4.1.2).

3.3.2 What to Fill: Candidate Proposal, Screening, and Verification

Learned proposals and first-order screening. After selecting the positions to re-mask, we perform one masked MGT forward pass. At each selected position i, we retain the top M codes under the MGT distribution as candidate replacements, restricting the search to discrete alternatives supported by the learned trajectory prior. Evaluating every proposal through full decoding is expensive, so we first screen the candidates using a Gumbel-Softmax relaxation. Let $\tilde{p}_{i,j}$ denote the Gumbel-Softmax weight of candidate j among the M proposals after adding Gumbel noise, and let $\mathbf{e}_{i,j}$ denote its codebook embedding. At each selected position, we form a probability-weighted relaxed embedding while keeping all other positions at their current hard embeddings. Then each candidate is scored using a first-order approximation of the geometric objective:

$$r_{i,j} = -\tilde{p}_{i,j} \frac{\partial J}{\partial \bar{\mathbf{e}}_i} \cdot (\mathbf{e}_{i,j} - \bar{\mathbf{e}}_i), \qquad \bar{\mathbf{e}}_i = \sum_{j=1}^{M} \tilde{p}_{i,j} \mathbf{e}_{i,j} \tag{2}$$

where $\bar{\mathbf{e}}_i$ is the relaxed embedding at position i, and $r_{i,j}$ is the first-order screening score for candidate j. The Gumbel-Softmax weight favors replacements supported by the learned prior, while the inner product favors candidates aligned with a local descent direction of J. We retain the top-k candidates according to $r_{i,j}$ for exact verification. While the gradient in Eq. 1 determines *where* the trajectory is edited, the gradient in Eq. 2 ranks *what* tokens are worth testing at those positions.

Verifier-guided sequential refinement. The first-order score is used only for screening, since it does not capture decoder nonlinearities or interactions between token replacements. We therefore verify shortlisted candidates using hard-token trajectories. Selected positions are visited in decreasing attribution order, and each candidate is substituted into the current token sequence, decoded, and evaluated with the exact objective J. For compactness, we write $\mathcal{D}(\mathbf{z})$ to denote decoding the codebook embeddings corresponding to token sequence $\mathbf{z}$. The best replacement is committed only if it improves the current solution:

$$z_i \leftarrow \begin{cases} \arg\min_{z_i' \in \mathcal{N}_i(\mathbf{z})} J\left(\mathcal{D}\left(\mathbf{z}^{(i \leftarrow z_i')}\right)\right), & \min_{z_i' \in \mathcal{N}_i(\mathbf{z})} J\left(\mathcal{D}\left(\mathbf{z}^{(i \leftarrow z_i')}\right)\right) < J(\mathcal{D}(\mathbf{z}))\,, \\ z_i, & \text{otherwise.} \end{cases} \tag{3}$$

where $\mathbf{z}^{(i \leftarrow z_i')}$ denotes the full sequence with position i replaced by z_i', and $\mathcal{N}_i(\mathbf{z})$ is the shortlist at that position. Accepted edits are applied immediately, so subsequent positions are evaluated against the updated trajectory. If none of the shortlisted candidates at the current position improves the solution, the corresponding position–token pairs are recorded as failed candidates and excluded from subsequent evaluation.

3.3.3 Iterative Refinement

At each round, we recompute how strongly each token contributes to the current geometric violation, producing a new re-masking set and MGT proposal pool. Candidate screening and sequential verification are then performed against the latest committed trajectory, so later rounds focus on the remaining violations. Refinement stops once the trajectory is feasible or the maximum number of rounds is reached. We then optionally apply only a few gradient steps of continuous optimization to the final token embeddings. Thus, discrete token replacement provides structural repair, while continuous polishing performs only local correction. Refinement is summarized in App. A.1.

4 Experiments

Evaluation setup. We organize the evaluation from controlled analysis of structural repair to broader generalization and real-world execution. *Ring Maze* tests whether MGMP can escape local planning basins that require substantial route changes (Sec. 4.1.1), while *Controlled Route Invalidation on Kuka* isolates GGTS-based repair from initial-generation quality by initializing all

methods from the same invalidated trajectory (Sec. 4.1.2). We then evaluate end-to-end planning on Maze2D, 7-DoF Kuka, and 14-DoF DualKuka under unseen obstacle layouts, additional obstacles, and unseen obstacle geometries (Sec. 4.2), followed by ablations (Sec. 4.3) and real-world Baxter experiments (Sec. 4.4). For the end-to-end benchmarks, we evaluate 1,000 planning problems across 50 unseen environments per task. Full environment construction, training-data generation, and evaluation protocols are provided in App. B.3.

Baselines and variants. We compare MGMP with eleven classical and learning-based planners: RRT* (Karaman & Frazzoli, 2011), P-RRT* (Qureshi & Ayaz, 2016), BIT* (Gammell et al., 2015), RMP (Ratliff et al., 2018), CHOMP (Ratliff et al., 2009), TrajOpt (Schulman et al., 2014), STOMP (Kalakrishnan et al., 2011), MπNets (Fishman et al., 2023), MPD (Carvalho et al., 2023), PBD (Luo et al., 2024), and SafeFlowMatcher (Yang et al., 2025). We further compare *MGT+EmbOpt (GD-30)*, which replaces GGTS with 30 steps of continuous embedding optimization, and *MGMP w/o Polish*, which removes the final five-step continuous polishing stage. See App. B.4 for details.

Metrics. For Ring Maze and end-to-end planning, we report *success rate (SR)*, *valid-sample fraction (VSF)*, *planning time (PT)*, and *running time (RT)*. Each generative planner produces 20 candidates per problem; SR measures whether at least one is valid, while VSF is the fraction of valid candidates. PT is the time to obtain a feasible solution, whereas RT measures the wall-clock cost of generating and refining the full candidate set. Details are provided in App.B.5. For *Controlled Route Invalidation*, we report *repair success rate (RS)* defined as the fraction of invalidated trajectories repaired into valid solutions. Among successful repairs, we additionally report the average and maximum joint-space deviation from the shared initial trajectory, together with the longest contiguous changed span. Full definitions are provided in App. B.7.

4.1 Escaping Local Planning Basins

We first use *Ring Maze* to test whether MGMP can escape local optima during planning that hard be resolved by local adjustments, and then use *Controlled Route Invalidation on Kuka* to isolate the contribution of GGTS-based trajectory repair from differences in initial trajectory generation.

4.1.1 Natural Basin Traps in Ring Maze.

We construct a compact 2-D Ring Maze benchmark with seven near-enclosing C-shaped obstacles, a standard geometry for inducing local planning minima. Their dense arrangement creates route-level traps, where escaping one enclosure may still require further route changes to reach the goal. Consequently, successful planning can require substantial route changes across the environment rather than local waypoint adjustments around individual collisions. To ensure sufficient coverage of such cases, we include start–goal pairs with states inside the C-shaped enclosures. We evaluate 4,000 problems across 100 unseen environments, of which 70% require escaping an enclosure. Environment, dataset, and implementation details are provided in Apps. B.3 and B.6.

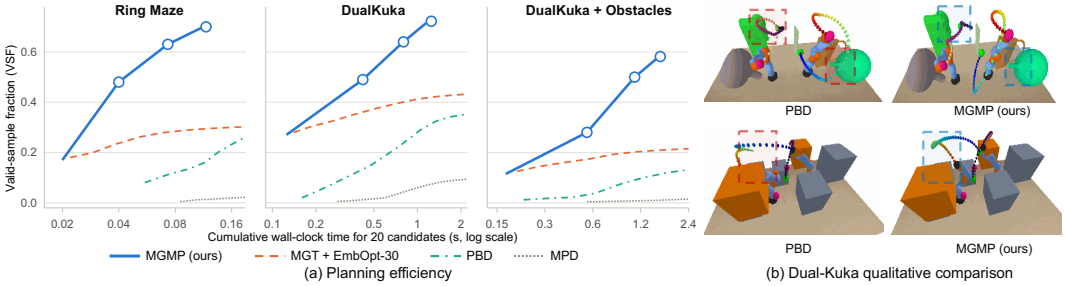
MGMP achieves the highest success rate (96%) among the evaluated refinement-based methods (Table 1). The improvement is particularly pronounced in valid-sample fraction, where MGMP reaches 67%, compared with 29% for MGT + EmbOpt and 23% for the best classical trajectory optimizer. MGMP w/o Polish already achieves 91% success and 64% valid-sample fraction, showing that GGTS provides most of the basin-escape capability, while the final polishing stage further improves feasibility. See Fig. 4(a) for representative Ring Maze trajectories. MGMP achieves these gains with only 0.08s runtime. Further analysis in App. C.1 shows that increasing the local-refinement budget does not close this gap: CHOMP changes little from 10–50 optimization iterations, while MPD improves only up to three guidance steps before degrading. As collision severity increases, PBD's repair rate drops sharply, whereas MGMP remains relatively stable, showing that GGTS is more capable of repairing deeply invalid trajectories that require substantial route changes.

4.1.2 Controlled Route Invalidation on Kuka

To isolate structural repair capability from initial-generation quality, we evaluate 500 invalidation problems across 50 unseen 7-DoF Kuka environments. For each problem, we first obtain a verified feasible Route A and pass it through the trajectory tokenizer to obtain a reconstructed route. We verify that this reconstructed route remains feasible in the original scene, and then relocate one obstacle to invalidate it while preserving a verified alternative Route B that is substantially separated from the invalidated Route A in joint space. All methods are initialized from the same reconstructed route

Table 1: Results on Ring Maze and Controlled Route Invalidation on Kuka. SR, VSF, RT, and RS denote success rate, valid-sample fraction, running time, and repair success rate, respectively. For Controlled Route Invalidation, trajectory-deviation statistics are reported over successful repairs.

Methods	Ring Maze			Controlled Route Invalidation on Kuka			
	SR↑	VSF↑	RT↓	RS↑	Avg. Dev.	Max. Dev.	Changed Span
CHOMP	0.73 ±0.09	0.23 ±0.10	0.56 ±0.14	0.46 ±0.15	0.82 ±0.19	1.30 ±0.32	31.20 ±9.40
TrajOpt	0.66 ±0.04	0.22 ±0.03	2.64 ±0.01	0.53 ±0.14	0.88 ±0.20	1.61 ±0.38	34.80 ±10.10
STOMP	0.23 ±0.01	0.05 ±0.03	0.45 ±0.07	0.57 ±0.15	0.96 ±0.22	1.66 ±0.35	39.60 ±10.80
MPD	0.14 ±0.06	0.02 ±0.02	0.21 ±0.06	0.38 ±0.13	0.83 ±0.20	1.43 ±0.35	31.90 ±9.60
PBD	0.67 ±0.10	0.26 ±0.09	0.16 ±0.05	0.44 ±0.14	0.93 ±0.22	1.65 ±0.39	37.20 ±10.30
SafeFlowMatcher	0.10 ±0.05	0.08 ±0.06	2.33 ±0.32	0.31 ±0.14	0.91 ±0.21	1.70 ±0.42	37.10 ±10.30
MGT+EmbOpt30	0.75 ±0.09	0.29 ±0.11	0.08 ±0.02	0.60 ±0.15	1.02 ±0.24	1.71 ±0.36	45.20 ±11.10
MGMP w/o Polish	0.91 ±0.06	0.64 ±0.13	**0.07** ±0.01	0.76 ±0.13	1.42 ±0.28	2.65 ±0.50	73.40 ±11.60
MGMP (Ours)	**0.96** ±0.04	**0.67** ±0.12	0.08 ±0.02	**0.82** ±0.12	1.38 ±0.27	2.40 ±0.44	71.70 ±11.20



(a) Planning efficiency (b) Dual-Kuka qualitative comparison

Figure 3: Planning efficiency and qualitative comparison. (a) Valid-sample fraction versus wall-clock time across three settings. (b) Dual-Kuka results under unseen obstacle geometries (top) and additional obstacles (bottom); PBD collides (red), while MGMP is collision-free (blue).

under the same modified constraints and directly apply only their respective refinement procedures, without generating a new initial trajectory. The modified scene therefore requires a structural change to the shared initial route rather than a local adjustment around the newly introduced collision. A representative example is shown in Fig. 4 (b). Initialization details are provided in App. B.7.

As shown in Table 1, MGMP achieves 82% repair success, outperforming MGT+EmbOpt (60%) and the strongest classical optimizer, STOMP (57%). MGMP w/o Polish already reaches 76%, showing that GGTS provides most of the improvement. Among MGMP's successful repairs, the longest changed span covers 71.7% of the trajectory, indicating that GGTS can modify extended portions of the route rather than only the immediate collision region. The similar repair magnitude of MGMP w/o Polish and full MGMP, together with the increase in repair success from 76% to 82% after polishing, supports complementary roles for the two stages: GGTS provides the large-scale trajectory repair, while polishing resolves residual local infeasibilities.

4.2 Generalization and Robustness

Generalization to Unseen Obstacle Layouts We first evaluate generalization to unseen obstacle layouts, where obstacle types and counts match the training environments, but obstacle placements and start-goal pairs are unseen (Table 2; implementation details in App. B.8). On Maze2D, MGMP achieves 100% success on both tasks, matching BIT* with lower planning time, while attaining the highest valid-sample fractions among learning-based planners. On 7-DoF Kuka, MGMP maintains a success rate comparable to BIT* while reducing planning time from 0.39 s to 0.13 s. It also improves the valid-sample fraction over PBD from 0.65 to 0.87. The advantage is more pronounced on the higher-dimensional 14-DoF DualKuka task: MGMP achieves a success rate of 0.96, compared with 0.85 for BIT*, 0.81 for CHOMP, and 0.79 for PBD, while reducing planning time from 2.01 s for BIT* to 0.39 s and achieving a valid-sample fraction of 0.70, substantially higher than PBD's 0.35. Overall, MGMP maintains high success rates on unseen layouts while producing more valid trajectories with lower runtime. Detailed results are provided in App. C.2.

Generalization to Additional-Obstacle Environments We next evaluate generalization to environments where scenes contain more obstacles than those observed during training. Additional setup details are provided in App. B.9. As shown in Table 3, MGMP remains effective as number of

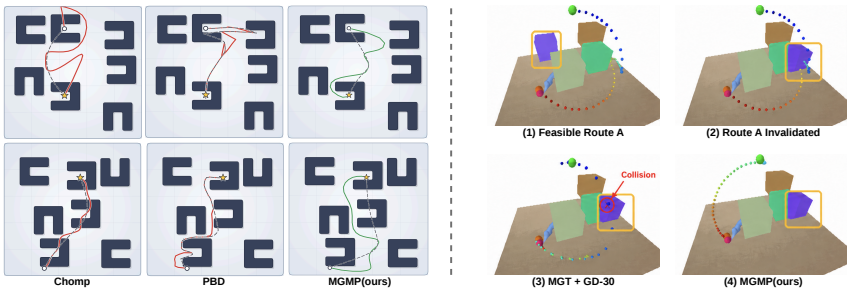


Figure 4: (a) Qualitative comparison on Ring Maze. Top and bottom rows show two planning problems. Dashed/solid curves denote initial/refined trajectories, and green/red denote collision-free/colliding solutions. (b) Example of Controlled Route Invalidation on Kuka.

Table 2: Results on unseen obstacle layouts. SR, VSF, and PT denote success rate, valid-sample fraction, and planning time in seconds. Results are reported as mean ± standard deviation.

Methods	**Maze2D-6**			**Concave Maze2D**			**7-DoF Kuka**			**14-DoF DualKuka**		
	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓
RMP	0.44 ±0.13	–	3.09 ±0.59	0.38 ±0.13	–	3.42 ±0.64	0.81 ±0.14	–	1.24 ±0.72	0.65 ±0.18	–	2.25 ±0.88
RRT*	0.99 ±0.01	–	0.54 ±0.13	0.98 ±0.03	–	1.24 ±0.22	0.64 ±0.17	–	2.31 ±0.78	0.44 ±0.17	–	3.49 ±0.64
P-RRT*	0.99 ±0.02	–	0.86 ±0.18	0.96 ±0.05	–	1.60 ±0.27	0.63 ±0.18	–	2.46 ±0.80	0.42 ±0.17	–	3.58 ±0.64
BIT*	**1.00** ±0.01	–	0.15 ±0.02	**1.00** ±0.01	–	0.29 ±0.04	**0.97** ±0.02	–	0.39 ±0.25	0.85 ±0.14	–	2.01 ±0.76
CHOMP	0.99 ±0.01	0.64 ±0.07	0.35 ±0.03	0.99 ±0.01	0.55 ±0.02	0.24 ±0.08	0.89 ±0.04	0.62 ±0.02	1.50 ±0.08	0.81 ±0.05	0.49 ±0.04	1.74 ±0.12
TrajOpt	0.99 ±0.01	0.87 ±0.01	0.47 ±0.02	0.99 ±0.01	0.72 ±0.07	1.91 ±0.02	0.67 ±0.03	0.25 ±0.06	2.41 ±0.01	0.67 ±0.08	0.25 ±0.06	3.95 ±0.23
STOMP	0.91 ±0.05	0.34 ±0.04	0.47 ±0.08	0.85 ±0.07	0.27 ±0.06	0.35 ±0.08	0.78 ±0.05	0.27 ±0.02	1.23 ±0.04	0.64 ±0.09	0.14 ±0.08	1.88 ±0.16
MπNets	0.98 ±0.01	–	0.13 ±0.09	0.94 ±0.02	–	0.14 ±0.02	0.85 ±0.08	–	0.23 ±0.01	0.61 ±0.01	–	0.74 ±0.03
MPD	0.83 ±0.10	0.31 ±0.04	0.14 ±0.03	0.59 ±0.11	0.30 ±0.04	0.27 ±0.08	0.62 ±0.28	0.35 ±0.12	0.35 ±0.03	0.28 ±0.21	0.09 ±0.09	0.73 ±0.09
PBD	**1.00** ±0.01	0.92 ±0.04	0.15 ±0.06	0.98 ±0.01	0.88 ±0.07	0.15 ±0.09	0.93 ±0.01	0.65 ±0.17	0.15 ±0.09	0.79 ±0.15	0.35 ±0.17	0.42 ±0.26
S.F.Matcher	0.50 ±0.03	0.29 ±0.05	1.86 ±0.12	0.45 ±0.01	0.32 ±0.01	2.00 ±0.16	0.45 ±0.06	0.28 ±0.03	2.47 ±0.02	0.29 ±0.04	0.13 ±0.01	2.86 ±0.05
MGMP (Ours)	**1.00** ±0.01	**0.98** ±0.01	**0.03** ±0.01	**1.00** ±0.01	**0.95** ±0.04	**0.07** ±0.01	**0.97** ±0.07	**0.87** ±0.14	**0.13** ±0.07	**0.96** ±0.06	**0.70** ±0.13	**0.39** ±0.12

obstacles increases. On 7-DoF Kuka, MGMP slightly improves success over BIT* (0.96 vs. 0.95) while reducing planning time from 0.73 s to 0.23 s, and increases the valid-sample fraction over CHOMP from 0.52 to 0.86. The advantage is larger on 14-DoF DualKuka, where MGMP improves success over BIT* from 0.63 to 0.84 and reduces planning time from 3.44 s to 0.90 s, while increasing the valid-sample fraction over CHOMP from 0.26 to 0.54. Taken together, these results indicate that MGMP is less sensitive to increased test-time constraint density, particularly in the higher-dimensional DualKuka setting, as further illustrated qualitatively in Fig. 3(b) and App. C.3.

Generalization to Unseen Obstacle Geometries We further evaluate generalization to unseen obstacle geometries by replacing the box obstacles used during training with ShapeNet (Chang et al., 2015) object meshes. We use ten everyday object categories and randomly sample four objects for each Kuka environment and five for each DualKuka environment. DeepSDF (Park et al., 2019) provides differentiable collision costs for the unseen meshes during refinement. Further details are provided in App. B.10. As shown in Table 3, On 7-DoF Kuka, it achieves 0.66 valid-sample fraction, compared with 0.47 for PBD. On 14-DoF DualKuka, MGMP achieves the highest success rate among all methods (0.87), compared with 0.83 for BIT* and 0.75 for PBD, while increasing the valid-sample fraction over CHOMP from 0.34 to 0.57. These results show that MGMP remains effective under substantial changes in obstacle geometry, as further illustrated in Fig. 3(b) and App. C.4.

Planning efficiency Figure 3 (a) compares the quality–time trade-off of different refinement strategies. MGMP starts from a strong initial candidate set and improves rapidly with only a few GGTS rounds, increasing valid-sample fraction from 0.18 to 0.67 on Ring Maze and from 0.28 to 0.70 on DualKuka. The same trend holds under additional obstacles, where MGMP reaches about 0.54 valid-sample fraction, compared with roughly 0.21 for MGT+EmbOpt-30 and 0.14 for PBD. These results show that MGMP benefits from both strong discrete initialization and fast geometry-guided repair, reaching substantially higher-quality candidate sets within a small inference budget.

4.3 Ablation Study

Table 3: Results on additional-obstacle and unseen-geometry environments. SR, VSF, and PT denote success rate, valid-sample fraction, and planning time in seconds, respectively.

Methods	Kuka Add.-Obs.			DualKuka Add.-Obs.			Kuka Unseen Geo.			DualKuka Unseen Geo.		
	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓	SR↑	VSF↑	PT↓
RMP	0.77 ±0.22	–	1.40 ±1.12	0.48 ±0.19	–	3.05 ±0.88	0.77 ±0.15	–	1.47 ±0.78	0.65 ±0.17	–	2.21 ±0.80
RRT*	0.65 ±0.23	–	2.24 ±1.03	0.30 ±0.16	–	4.01 ±0.56	0.60 ±0.15	–	2.58 ±0.73	0.46 ±0.19	–	3.44 ±0.74
P-RRT*	0.66 ±0.23	–	2.38 ±1.02	0.29 ±0.17	–	4.14 ±0.54	0.57 ±0.15	–	2.84 ±0.70	0.43 ±0.18	–	3.58 ±0.70
BIT*	0.95 ±0.10	–	0.73 ±0.62	0.63 ±0.21	–	3.44 ±0.81	**0.91** ±0.10	–	0.99 ±0.58	0.83 ±0.19	–	1.88 ±0.96
CHOMP	0.77 ±0.02	0.52 ±0.04	2.16 ±0.05	0.68 ±0.07	0.26 ±0.06	2.95 ±0.01	0.65 ±0.02	0.33 ±0.01	2.20 ±0.06	0.57 ±0.08	0.34 ±0.05	2.68 ±0.08
TrajOpt	0.68 ±0.02	0.27 ±0.01	2.35 ±0.01	0.60 ±0.01	0.11 ±0.03	3.80 ±0.28	0.63 ±0.05	0.37 ±0.02	2.35 ±0.18	0.74 ±0.03	0.31 ±0.07	3.34 ±0.39
STOMP	0.73 ±0.01	0.28 ±0.06	1.18 ±0.06	0.41 ±0.09	0.07 ±0.06	1.70 ±0.19	0.66 ±0.02	0.23 ±0.02	1.38 ±0.15	0.57 ±0.01	0.18 ±0.06	1.38 ±0.15
MπNets	0.52 ±0.06	–	0.42 ±0.03	0.29 ±0.09	–	1.97 ±0.11	–	–	–	–	–	–
MPD	0.37 ±0.27	0.08 ±0.10	0.46 ±0.04	0.12 ±0.11	0.01 ±0.03	1.80 ±0.35	0.34 ±0.11	0.14 ±0.17	0.68 ±0.02	0.30 ±0.04	0.11 ±0.06	1.98 ±0.09
PBD	0.76 ±0.17	0.50 ±0.19	0.47 ±0.08	0.51 ±0.22	0.12 ±0.10	1.33 ±0.19	0.76 ±0.19	0.47 ±0.19	0.23 ±0.02	0.75 ±0.19	0.36 ±0.19	0.96 ±0.08
S.F.Matcher	0.38 ±0.09	0.28 ±0.03	2.46 ±0.05	0.20 ±0.03	0.16 ±0.02	1.87 ±0.05	0.31 ±0.07	0.21 ±0.01	2.47 ±0.03	0.33 ±0.03	0.17 ±0.02	2.92 ±0.01
MGMP (Ours)	**0.96** ±0.05	**0.86** ±0.14	**0.23** ±0.12	**0.84** ±0.09	**0.54** ±0.16	**0.90** ±0.21	0.90 ±0.08	**0.66** ±0.13	**0.21** ±0.10	**0.87** ±0.09	**0.57** ±0.14	**0.72** ±0.18

Table 4 isolates the main design choices in GGTS. For *where to edit*, confidence-only remasking reduces SR to 0.84/0.92 on Ring Maze/Kuka, while geometry-only remasking reaches 0.93/0.95, showing that geometric attribution provides the primary localization signal, with confidence serving as complementary guidance. For *what to fill* and *how to commit*, probability-only screening and independent exact search reach 0.88/0.94 and 0.91/0.95 SR, respectively, below Full MGMP (0.96/0.97), supporting geometry-guided screening and sequential verification. Finally, MGT+EmbOpt-30 reaches only 0.75/0.90 SR, whereas MGMP w/o Polish already achieves 0.91/0.94. Polishing further improves this to 0.96/0.97, indicating that discrete search performs the main structural repair while continuous optimization resolves residual local infeasibilities. Additional analyses in App. C.5 examine the trade-off between first-order screening and exact verification, and sensitivity to the main refinement hyperparameters.

Table 4: Ablation results on Ring Maze and 7-DoF Kuka. SR, VSF, and RT denote success rate, valid-sample fraction, and runtime.

Variant	Ring Maze			7-DoF Kuka		
	SR ↑	VSF ↑	RT ↓	SR ↑	VSF ↑	RT ↓
MGT (Conf.-Refine)	0.69	0.20	0.03	0.84	0.58	0.07
MGT + EmbOpt-30	0.75	0.29	0.08	0.90	0.74	0.30
Conf.-only Remask	0.84	0.49	0.08	0.92	0.78	0.43
Geom.-only Remask	0.93	0.64	0.08	0.95	0.84	0.44
Prob.-only Screening	0.88	0.55	0.07	0.94	0.81	0.41
Independent Exact Search	0.91	0.60	0.08	0.95	0.82	0.44
MGMP w/o Polish	0.91	0.64	0.07	0.94	0.83	0.40
Full MGMP	**0.96**	**0.67**	**0.08**	**0.97**	**0.87**	**0.44**

4.4 Real World Experiments

We conduct real-world tabletop experiments with a Baxter robot on three downstream tasks: shelf placing, button pressing, and pen drawing. The planner takes the start and goal configurations and obstacle locations as input and outputs a joint-space trajectory. We train a task-agnostic planner from demonstrations collected with Baxter's left arm while avoiding four tracked tabletop box obstacles. At test time, we evaluate five unseen environments with three planning problems each, using five candidate trajectories per method. For the 3D tasks, two training boxes are replaced with larger unseen rectangular obstacles. Table 5 shows that MGMP performs consistently well across all tasks. See details in App. D.



Figure 5: Real-world tasks: (a) button pressing; (b) shelf placing; (c) drawing.

Table 5: Results on real-world tasks.

Method	Shelf		Button		Drawing	
	SR	VSF	SR	VSF	SR	VSF
MPD	0.533	0.293	0.733	0.493	0.733	0.333
PBD	0.400	0.213	0.667	0.360	0.600	0.267
MGMP	**0.733**	**0.360**	**0.800**	**0.613**	**0.867**	**0.453**

5 Conclusion

We presented MGMP, which reframes generative motion planning by keeping the learned trajectory prior active beyond initial generation. A shared discrete trajectory space supports both efficient parallel generation and structural repair, while GGTS turns prior-supported motion alternatives into a geometry-guided, targeted search space. Rather than relying on local continuous refinement to discover new routes implicitly, MGMP directly restructures trajectories through discrete token replacement, using continuous optimization only for residual correction. Controlled basin-escape and route-invalidation experiments show that discrete search provides the main repair capability, while broader evaluations demonstrate generalization across diverse planning settings. More broadly, MGMP shows how learned trajectory priors can serve not only as generators, but also as structured proposal spaces for geometry-guided test-time adaptation.

AI USE STATEMENT

We used generative AI tools solely for language editing, including improving the grammar, clarity, and phrasing of the paper. All AI-assisted edits were reviewed by the authors for technical accuracy and consistency with the intended meaning. Generative AI was not used to generate experimental results or determine the scientific conclusions of this work. The authors take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

Experiment details, including all hyperparameters and implementation details, are in Appendix Sec. B, with further results in Appendix Sec. C. These materials enable full reproducibility.

REFERENCES

Lukas Lao Beyer and Sertac Karaman. Motion planning in compressed representation spaces. *arXiv preprint arXiv:2606.30940*, 2026.

Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1916–1923. IEEE, 2023.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11315–11325, 2022.

Huiwen Chang, Han Zhang, Jarred Barber, Aaron Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Patrick Murphy, William T Freeman, Michael Rubinstein, et al. Muse: Text-to-image generation via masked generative transformers. In *International Conference on Machine Learning*, pp. 4055–4075. PMLR, 2023.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

Dibyendu Das, Yuanjie Lu, Erion Plaku, and Xuesu Xiao. Motion memory: Leveraging past experiences to accelerate future motion planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16467–16474. IEEE, 2024.

Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *conference on Robot Learning*, pp. 967–977. PMLR, 2023.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 3067–3074. IEEE, 2015.

Chuan Guo, Yuxuan Mu, Muhammad Gohar Javed, Sen Wang, and Li Cheng. Momask: Generative masked modeling of 3d human motions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1900–1910, 2024.

Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 2951–2957. IEEE, 2015.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pp. 9902–9915. PMLR, 2022.

Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=cA77NrVEuqn.

Jacob J Johnson, Ahmed H Qureshi, and Michael C Yip. Learning sampling dictionaries for efficient and generalizable robot motion planning with transformers. *IEEE Robotics and Automation Letters*, 8(12):7946–7953, 2023.

Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pp. 4569–4574. IEEE, 2011.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics Autom.*, 12(4): 566–580, 1996. doi: 10.1109/70.508439. URL https://doi.org/10.1109/70.508439.

Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. In *International Conference on Learning Representations*, volume 2025, pp. 81979–82026, 2025.

Yunhao Luo, Chen Sun, Joshua B. Tenenbaum, and Yilun Du. Potential based diffusion motion planning. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 33486–33510. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/luo24h.html.

Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 9–15. IEEE, 2016.

Andreas Orthey, Constantinos Chamzas, and Lydia E. Kavraki. Sampling-based motion planning: A comparative review. *Annu. Rev. Control. Robotics Auton. Syst.*, 7(1), 2024. doi: 10.1146/ANNUREV-CONTROL-061623-094742. URL https://doi.org/10.1146/annurev-control-061623-094742.

Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5190. IEEE, 2014.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 165–174, 2019.

Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling. *CoRR*, abs/2502.03540, 2025. doi: 10.48550/ARXIV.2502.03540. URL https://doi.org/10.48550/arXiv.2502.03540.

Ekkasit Pinyoanuntapong, Pu Wang, Minwoo Lee, and Chen Chen. Mmm: Generative masked motion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1546–1555, 2024.

Ahmed H Qureshi and Michael C Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6582–6588. IEEE, 2018.

Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079–1093, 2016.

Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Trans. Robotics*, 37(1):48–66, 2021. doi: 10.1109/TRO.2020.3006716. URL https://doi.org/10.1109/TRO.2020.3006716.

Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE international conference on robotics and automation*, pp. 489–494. IEEE, 2009.

Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pp. 1–10. Berlin, Germany, 2013.

John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

Mingyo Seo, Yoonyoung Cho, Yoonchang Sung, Peter Stone, Yuke Zhu, and Beomjoon Kim. Presto: Fast motion planning using diffusion models based on key-configuration environment representation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10861–10867. IEEE, 2025.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Kehan Wen, Yutong Hu, Yao Mu, and Lei Ke. Mˆ 3pc: Test-time model predictive control using pretrained masked trajectory model. In *International Conference on Learning Representations*, volume 2025, pp. 37721–37743, 2025.

Philipp Wu, Arjun Majumdar, Kevin Stone, Yixin Lin, Igor Mordatch, Pieter Abbeel, and Aravind Rajeswaran. Masked trajectory models for prediction, representation, and control. In *International Conference on Machine Learning*, pp. 37607–37623. PMLR, 2023.

Jun Yamada, Chia-Man Hung, Jack Collins, Ioannis Havoutis, and Ingmar Posner. Leveraging scene embeddings for gradient-based motion planning in latent space. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5674–5680. IEEE, 2023.

Jeongyong Yang, Seunghwan Jang, and Soojean Han. Safeflowmatcher: Safe and fast planning using flow matching with control barrier functions. *CoRR*, abs/2509.24243, 2025. doi: 10.48550/ARXIV.2509.24243. URL https://doi.org/10.48550/arXiv.2509.24243.

Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. *Advances in Neural Information Processing Systems*, 34:4274–4289, 2021.

Lipeng Zhuang, Shiyu Fan, Florent P Audonnet, Yingdong Ru, Edmond SL Ho, Gerardo Aragon Camarasa, and Paul Henderson. Masked generative policy for robotic control. *arXiv preprint arXiv:2512.09101*, 2025.

Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10): 1164–1193, 2013.

Appendices

A Additional Method Details **14**

A.1 MGMP Inference with Geometry-Guided Token Search 14

A.2 Planning Objectives for Different Task Settings 14

B Experiment Details **15**

B.1 Hyperparameters 15

B.2 Network Architecture 17

B.3 Simulation Dataset and Training Data 18

B.4 Baseline Details 18

B.5 Running time 19

B.6 Experiment Details for Ring Maze 19

B.7 Implementation Details for Controlled Route Invalidation on Kuka 20

B.8 Implementation Details for Unseen Obstacle Layouts 20

B.9 Generalization to Additional-Obstacle Environments Setup 21

B.10 Generalization to Unseen Obstacle Geometries Environments Setup 21

C Additional Simulation Results **23**

C.1 Further Analysis on Ring Maze 23

C.2 Generalization to Unseen Obstacle Layouts 23

C.3 Generalization to Additional-Obstacle Environments 24

C.4 Generalization to Unseen Obstacle Geometries 24

C.5 Ablation Study And Hyperparameter Experiments 25

D Real-World Experiments **26**

D.1 Environment setup 26

D.2 Qualitative results 26

A Additional Method Details

A.1 MGMP Inference with Geometry-Guided Token Search

Algorithm 1 MGMP Inference with Geometry-Guided Token Search

Require: Condition $\mathbf{c} = (c_s, c_g, w)$, MGT f_θ, decoder $\mathcal{D}$, codebook $\mathcal{K}$, objective J, Tabu-list $\mathcal{T}$
Require: Rounds R, proposal size M, shortlist size k, attribution mass ρ, polishing steps P, momentum μ
Ensure: Refined trajectory $\hat{\mathbf{x}}$

▷ Initial masked generation

$\boldsymbol{\ell} \leftarrow f_\theta([\textsc{Mask}]^N, \mathbf{c}), \quad \mathbf{p} \leftarrow \mathrm{softmax}(\boldsymbol{\ell})$
$\mathbf{z} \leftarrow \mathrm{GumbelMax}(\boldsymbol{\ell}), \quad c_i \leftarrow p_{i,z_i}$
$\mathbf{e} \leftarrow \mathcal{K}[\mathbf{z}], \quad \hat{\mathbf{x}} \leftarrow \mathcal{D}(\mathbf{e}), \quad \mathcal{T} \leftarrow \emptyset$

▷ Geometry-Guided Token Search

for $r = 1$ to R **do**
  if $\hat{\mathbf{x}}$ is feasible **then**
    break
  end if

▷ WHERE to edit

  $\mathbf{e} \leftarrow \mathcal{K}[\mathbf{z}]$
  $s_i \leftarrow \|\nabla_{\mathbf{e}_i} J(\mathcal{D}(\mathbf{e}))\|_2 \big(1 + \lambda_{\mathrm{conf}}(1 - c_i)\big)$
  $\mathcal{I} \leftarrow \mathrm{CumulativeTop}(s, \rho) \cup \mathrm{CollisionTokens}(\hat{\mathbf{x}}, w)$

▷ WHAT to fill

  $\boldsymbol{\ell} \leftarrow f_\theta(\mathrm{Mask}(\mathbf{z}, \mathcal{I}), \mathbf{c}), \quad \mathbf{p} \leftarrow \mathrm{softmax}(\boldsymbol{\ell})$
  for all $i \in \mathcal{I}$ **do**
    $c_i \leftarrow p_{i,z_i}, \quad \mathcal{C}_i \leftarrow \mathrm{TopM}(p_i)$
    $\tilde{\mathbf{p}}_i \leftarrow \mathrm{softmax}\Big(\frac{\boldsymbol{\ell}_i[\mathcal{C}_i] + \mathbf{g}_i}{\tau}\Big), \quad \mathbf{g}_i \sim \mathrm{Gumbel}(0, 1)$
    $\bar{\mathbf{e}}_i \leftarrow \sum_{j \in \mathcal{C}_i} \tilde{p}_{i,j} \mathcal{K}[j]$
    $\mathcal{S}_i \leftarrow \mathrm{TopK}_{j \in \mathcal{C}_i} \left[-\tilde{p}_{i,j} \nabla_{\bar{\mathbf{e}}_i} J \cdot (\mathcal{K}[j] - \bar{\mathbf{e}}_i)\right]$
  end for

▷ Exact sequential verification

  for all $i \in \mathrm{SortDesc}(\mathcal{I}, s_i)$ **do**
    $j^\star \leftarrow \arg\min_{\substack{j \in \mathcal{S}_i \\ (i,j) \notin \mathcal{T}}} J\big(\mathcal{D}(\mathcal{K}[\mathbf{z}^{(i \leftarrow j)}])\big)$
    if $J(\mathcal{D}(\mathcal{K}[\mathbf{z}^{(i \leftarrow j^\star)}])) < J(\hat{\mathbf{x}})$ **then**
      $z_i \leftarrow j^\star, \quad \hat{\mathbf{x}} \leftarrow \mathcal{D}(\mathcal{K}[\mathbf{z}]), \quad \mathcal{T} \leftarrow \emptyset$
    else
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(i, j) : j \in \mathcal{S}_i\}$
    end if
  end for
end for

▷ Optional continuous polishing

$\mathbf{e} \leftarrow \mathcal{K}[\mathbf{z}], \quad \mathbf{m} \leftarrow \mathbf{0}$
for $t = 1$ to P **do**
  $\mathbf{m} \leftarrow \mu \mathbf{m} + \nabla_{\mathbf{e}} J(\mathcal{D}(\mathbf{e}))$
  $\mathbf{e} \leftarrow \mathbf{e} - \mathbf{m}$
end for
return $\mathcal{D}(\mathbf{e})$

A.2 Planning Objectives for Different Task Settings

After training, our model can handle several general planning scenarios.

Single-agent planning. In this scenario, the configuration x is from a single point or a single robotic arm. Generative planning only needs to consider the start/end distance error and scene collisions over the whole length T of the trajectory. The objective function is defined as follows:

$$\mathcal{L} = \sum_{t=1}^{T} \lambda_c \mathcal{L}_{\mathcal{SDF}}(\hat{x}_t, w) + \lambda_{sg} \mathcal{L}_{sg}(\hat{x}, c_s, c_g) \tag{4}$$

where λ_* are weight factors for the start/goal error $\mathcal{L}_{sg}$ and collision error $\mathcal{L}_{\mathcal{SDF}}$. The former is an L1 loss on configurations, while the latter is given by $\mathcal{L}_{\mathcal{SDF}} = \mathrm{ReLU}(-\mathrm{SDF}(\hat{x}, w) + \epsilon)$ where ϵ defines a margin around objects.

Dual-agent planning. For dual-agent or dual-arm planning tasks, in addition to avoiding collisions with the environment, it is also necessary to prevent collisions between the two agents/arms. Accordingly, a self-collision term is introduced to the optimization objective:

$$\mathcal{L} = \sum_{t=1}^{T} (\lambda_c \mathcal{L}_{\mathcal{SDF}}(\hat{x}_t^0, w) + \lambda_c \mathcal{L}_{\mathcal{SDF}}(\hat{x}_t^1, w) + \lambda_s \mathcal{L}_{\mathcal{SDF}}(\hat{x}_t^0, \hat{x}_t^1)) + \lambda_{sg} \mathcal{L}_{sg} \tag{5}$$

where λ_s is the weight for self-collision error, $\hat{x}_t^0$ and $\hat{x}_t^1$ are the trajectory of each agent at frame t.

Planning with additional obstacles. Our model can perform motion planning with a larger number of obstacles than seen in its training data. We partition the set of obstacles $\{w\}$ into J subsets, where the cardinality of each does not exceed the maximum number seen by the obstacle encoder during training. The problem is then reformulated as solving over the joint solution space conditioned on separate obstacle sets. The model conditioned on each subset can be viewed as an expert over those obstacles, and the trajectory constraints for the full set are given by multiplying these, similar to a product of experts: $e = \sum_j^J e_j$, $p(z \mid \{w\}) \propto \prod_{j=1}^J p(z \mid w_j)$ Based on this, we predict logits for each set individually and sum them to obtain the final global-condition logits. These logits are then sent into the softmax operator to calculate the final probability distribution and the codes. The optimization objective for the codes is the same as in the single-agent task.

B Experiment Details

B.1 Hyperparameters

We report the hyperparameters used for MGMP on Maze2D, 7-DoF Kuka, and 14-DoF DualKuka. Ring Maze uses the same configuration as Maze2D, while Controlled Route Invalidation uses the same configuration as the 7-DoF Kuka experiments. No additional hyperparameter tuning is performed on Ring Maze or Controlled Route Invalidation. MGMP consists of a VQ-VAE trajectory tokenizer, a masked generative transformer, and test-time token-space refinement followed by short continuous polishing. Tables 6, 7, and 8 summarize the corresponding hyperparameters.

Table 6: VQ-VAE hyperparameters.

Parameter	**Maze2D**	**7-DoF Kuka**	**14-DoF DualKuka**
Codebook size	8192	8192	8192
Code dimension	16	256	256
Encoder width	16	256	256
Encoder depth	4	4	4
Downsampling rate	2	2	1
Stride size	2	2	2
Dilation growth rate	3	3	3
Activation	SiLU	SiLU	SiLU
Quantizer	ema_reset	ema_reset	ema_reset
EMA decay μ	0.95	0.95	0.95
Reconstruction loss	smooth-ℓ_1	smooth-ℓ_1	smooth-ℓ_1
Commit weight	0.02	0.02	0.02
Velocity loss weight	0.5	0.5	0.5
FK loss weight	-	0.6	0.6
Learning rate	3×10^{-4}	3×10^{-4}	1×10^{-4}
LR milestones	[250k, 400k, 600k]	[250k, 400k, 600k]	[250k, 400k, 600k]
LR decay γ	0.1	0.1	0.1
Warmup iterations	1000	1000	1000
Weight decay	0.0	0.0	1×10^{-6}
Training iterations	800k	800k	800k
Batch size	256	256	256

Table 7: Masked generative transformer hyperparameters.

Parameter	**Maze2D**	**7-DoF Kuka**	**14-DoF DualKuka**
Embedding dimension	256	512	512
Number of layers	8	8	8
Number of heads	16	16	16
FFN multiplier	4	4	4
Dropout	0.2	0.2	0.2
Block size	16	16	32
Learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Weight decay	1×10^{-6}	1×10^{-4}	1×10^{-4}
Optimizer	AdamW	AdamW	AdamW
Training iterations	200k	200k	200k
Batch size	256	256	256

Table 8: Test-time GGTS and continuous polishing hyperparameters. During verification, the number of remasked tokens is capped at 40% of the trajectory tokens; for the Kuka setting, this corresponds to at most four tokens. Continuous polishing uses momentum-accumulated raw gradients with no separate learning rate. The SDF and SG weights are switched to their post-convergence values once the SDF loss falls below 5×10^{-4}.

Parameter	**Maze2D**	**7-DoF Kuka**	**14-DoF DualKuka**
GGTS refinement rounds	2	2	2
Token updates per round	2	2	2
Proposed code candidates per update	64	16	16
Candidates retained after screening	32	2	2
Maximum verification remask ratio	0.4	0.4	0.4
Polishing iterations	5	5	5
Momentum	0.9	0.9	0.9
SDF weight, pre-convergence	200	300	300
SG weight, pre-convergence	100	150	150
SDF weight, post-convergence	400	600	600
SG weight, post-convergence	100	450	450

For the standard Maze2D, Kuka, and DualKuka environments, we randomly split the dataset into training and validation sets with a 9:1 ratio. All hyperparameters are selected by grid search on the corresponding held-out validation set for each benchmark. For the VQ-VAE trajectory tokenizer, we choose the configuration that achieves the lowest validation reconstruction error. For the masked generative transformer and test-time refinement, we select the configurations that achieve the highest validation success rate.

B.2 Network Architecture

MGMP consists of two trainable network components: a VQ-VAE trajectory tokenizer and a masked generative transformer. The tokenizer learns a compact discrete representation of continuous trajectories, while the transformer models the conditional distribution over trajectory tokens given the start configuration, goal configuration, and obstacle representation.

Trajectory tokenizer. The trajectory tokenizer is implemented as a 1D-CNN VQ-VAE with an encoder–decoder architecture. The encoder is composed of 3 temporal convolutional blocks, each consisting of a 1D-CNN layer and a residual 1D-CNN layer. The kernel size of the 1D-CNN layer is 4. The decoder shares the mirrored structure with the encoder. The EMA tokenizer is selected as the tokenizer for the trajectory codes. The temporal downsampling factor for the single-agent task is $N/T = 1/4$, which means four trajectory steps will be encoded into 1 token. For the dual-agent task, this ratio is set as $1/2$ for better reconstruction results. Objective is:

$$\mathcal{L} = \lambda_{\text{rec}}\mathcal{L}_{\text{smoothL1}} + \beta\|\hat{\mathbf{e}} - \text{sg}[\mathbf{e}]\|_2^2 + \lambda_v\mathcal{L}_{\text{vel}} + \lambda_{\text{FK}}\mathcal{L}_{\text{FK}}.$$

The forward-kinematics loss is applied only to the Kuka and DualKuka trajectories; for Maze2D, $\lambda_{\text{FK}} = 0$.

Masked generative transformer. The masked generative transformer consists of two components. The first is a lightweight obstacle encoder that takes the obstacle center coordinates $w_C \in \mathbb{R}^{n\times3}$ as input. This encoder is built on 3-layer self-attention blocks without positional embedding. The hidden dimension and the output dimension are 256.

The masked generative transformer (MGT) backbone consists of 8 self-attention-based Transformer blocks. The MGT hidden dimension is 256 for Maze2D and 512 for Kuka/DualKuka. The start/goal condition is projected by a 2-layer MLP. The obstacle embedding is injected into the transformer through an AdaLN block. We apply conditional modulation only to the first six Transformer layers, while the final two layers serve as the output head.

B.3 Simulation Dataset and Training Data

We use BIT* (Gammell et al., 2015) to generate expert trajectories for all simulated environments using Pybullet simulation (Coumans & Bai, 2016). Each training sample is a feasible solution connecting a randomly sampled start-goal pair under a fixed obstacle layout. For each task, we construct 500 training environments and generate 400 trajectories per environment, resulting in 200K trajectory samples per task. We developed our task settings from (Luo et al., 2024). All training trajectories are collision-free BIT* (Gammell et al., 2015) solutions. Evaluation environments and start-goal pairs are disjoint from the training set.

Ring Maze Each Ring Maze environment is a $5 \times 5\,\mathrm{m}$ 2-D workspace containing seven near-enclosing C-shaped obstacles, each with a single opening oriented along one of the four cardinal directions. This construction creates enclosed regions with narrow exits that may require substantial route changes, while the obstacle configuration remains fixed throughout each planning problem. We generate 500 training, 20 validation, and 100 unseen test environments. Each training environment contains 400 BIT*-generated expert planning problems, yielding 200K training trajectories. We sample 10 problems per validation environment and 40 per test environment, resulting in 200 validation and 4,000 test problems. Because uniform start–goal sampling rarely places endpoints inside the ring cavities, we use a trap-aware sampler covering *ring-to-ring*, *ring-to-free*, *free-to-ring*, and *free-to-free* queries, where "ring" denotes a collision-free state inside a C-shaped enclosure and "free" denotes a state in the remaining free workspace. Under this protocol, 70% of the test problems require escaping at least one enclosure. BIT* is used to generate the training trajectories; start and goal states are checked for collision validity, and only problems for which BIT* finds a feasible solution are retained.

Maze2D. Maze2D is a low-dimensional planning task in which a holonomic point robot moves in a two-dimensional workspace. The robot state is its planar position, and the planner must generate a collision-free path from a start state to a goal state while avoiding square or concave obstacles. Collision is determined by whether the point robot lies inside an obstacle region. The dataset contains feasible paths generated by BIT* (Gammell et al., 2015) under randomly sampled obstacle layouts and start-goal pairs.

Kuka. Kuka evaluates joint-space motion planning for a 7-DoF manipulator in cluttered 3D workspaces. Each training environment contains randomly placed box obstacles around the robot. For each environment, BIT* (Gammell et al., 2015) generates feasible joint-space trajectories between randomly sampled start and goal configurations. The resulting trajectories are used to train the trajectory tokenizer and masked generative transformer.

DualKuka. DualKuka extends the Kuka setting to two coupled 7-DoF manipulators operating in a shared workspace. The state dimension is therefore 14, and feasible trajectories must avoid both workspace obstacles and inter-arm collisions. As in the Kuka setting, each environment contains randomly placed obstacles, and BIT* (Gammell et al., 2015) is used to generate feasible trajectories for randomly sampled start-goal pairs.

Trajectory-level training samples. We preserve trajectory boundaries when constructing training samples. Unlike stride-one sliding-window augmentation over concatenated trajectories, our training samples are extracted from individual BIT* (Gammell et al., 2015) solutions. This avoids windows that cross different planning problems and therefore do not correspond to a valid trajectory for a single start-goal pair. Each training sequence is thus consistent with its own start configuration, goal configuration, and obstacle layout.

B.4 Baseline Details

We compare MGMP with eleven representative baselines, covering both classical and learning-based planning methods.

Classical planners. RRT* (Karaman & Frazzoli, 2011), P-RRT* (Qureshi & Ayaz, 2016), and BIT* (Gammell et al., 2015) are sampling-based planners with different exploration strategies; BIT*

additionally combines informed sampling with graph search to progressively improve candidate paths. RMP (Ratliff et al., 2018) is a reactive motion policy that combines goal-reaching and obstacle-avoidance objectives online. We further include three trajectory-optimization baselines: CHOMP (Ratliff et al., 2009), which performs gradient-based optimization of collision and smoothness objectives; TrajOpt (Schulman et al., 2014), which optimizes trajectories through sequential convex optimization; and STOMP (Kalakrishnan et al., 2011), which uses stochastic trajectory perturbations to minimize planning costs. All classical planners are given a maximum planning budget of 5,s per problem.

Learning-based planners. MπNets (Fishman et al., 2023) learns a neural motion policy conditioned on the planning goal and environment. Motion Planning Diffusion (MPD) (Carvalho et al., 2023) models a trajectory prior with diffusion and incorporates planning costs during inference to guide sampled trajectories. Potential-Based Diffusion (PBD) (Luo et al., 2024) combines diffusion generation with potential-based guidance for goal reaching and obstacle avoidance. SafeFlowMatcher (Yang et al., 2025) uses flow matching for trajectory generation and a prediction–correction procedure with control barrier functions to continuously correct generated paths toward the safe set.

B.5 Running time

All experiments are run on a single workstation with an AMD Ryzen 9 5950X CPU and one NVIDIA RTX 4090 GPU. Device assignment follows the implementation of each planner: sampling-based planners (RRT*, P-RRT*, and BIT*) use CPU implementations based on OMPL, while learning-based planners and trajectory-optimization baselines use GPU. Candidate-based methods are evaluated with 20 trajectories per planning problem. Regardless of the internal collision objective, guidance mechanism, or optimizer used by a method, all returned trajectories are certified using the same final collision checker and endpoint-validity criteria. Sampling-based planners return a single solution and therefore do not report valid-sample fraction.

Planning time (PT) measures the wall-clock time from planner invocation until the first trajectory is certified as feasible, or until the maximum planning budget is reached. It includes test-time sampling, optimization or guidance, collision-distance and forward-kinematics evaluations where applicable, and the validity checks required to certify the returned solution. Running time (RT), in contrast, measures the wall-clock cost of generating and refining the complete set of 20 candidate trajectories. One-time model and dataset loading and the post-hoc validation used solely to compute valid-sample fraction are excluded from both timing measurements. Results are reported as mean $\pm$ standard deviation across test environments, after aggregating the planning queries within each environment.

B.6 Experiment Details for Ring Maze

For Ring Maze, all candidate-based methods are evaluated using 20 trajectories per planning problem under the common validity criteria to compare with learning-based motion planners. For the trajectory-optimization baselines CHOMP (Ratliff et al., 2009), TrajOpt (Schulman et al., 2014), and STOMP (Kalakrishnan et al., 2011), we additionally control the trajectory initialization. All three methods receive the same set of 20 initial trajectories for each query: one straight-line configuration-space interpolation between the start and goal and 19 randomized cubic Bézier trajectories with the same fixed endpoints. The two interior Bézier control points are sampled within the workspace bounds using deterministic per-query random seeds and do not depend on the obstacle configuration. The initial trajectories use the same waypoint discretization. CHOMP, TrajOpt, and STOMP then independently refine this common candidate set using their respective optimization procedures. For STOMP, the internal stochastic rollouts used to estimate an update are not counted as additional planning candidates. Method-specific hyperparameters for the trajectory optimizers are selected on held-out validation environments that are disjoint from the test environments, under the same 5 s maximum planning budget used at test time, and are fixed during evaluation. MPD (Carvalho et al., 2023), PBD (Luo et al., 2024), and SafeFlowMatcher (Yang et al., 2025) each generate 20 candidates using their native sampling procedures and apply their respective test-time refinement mechanisms. For the MGMP variants, MGT+EmbOpt-30 uses the same MGT-generated candidates as MGMP but replaces discrete token refinement with 30 steps of continuous embedding-space optimization. MGMP w/o Polish applies GGTS without the final continuous polishing stage, while full MGMP applies GGTS followed by the short continuous polishing stage.

B.7 Implementation Details for Controlled Route Invalidation on Kuka

For each controlled route-invalidation problem, we first obtain a verified feasible trajectory x_A (Route A). To remove representation mismatch between continuous- and token-based refinement methods, we encode x_A with the MGMP tokenizer and decode the resulting hard-token sequence z_A to obtain a common reconstructed trajectory

$$\tilde{x}_A = D(E(x_A)).$$

We retain only cases for which $\tilde{x}_A$ remains feasible in the original scene. We then relocate one obstacle to invalidate $\tilde{x}_A$ while preserving an alternative feasible route.

All methods receive the same invalidated trajectory $\tilde{x}_A$ and the same modified obstacle configuration. CHOMP, TrajOpt, STOMP, and SafeFlowMatcher-Corr. are initialized directly from $\tilde{x}_A$, with the prediction stage of SafeFlowMatcher bypassed.

For PBD, we use its native trajectory-refinement procedure rather than fresh generation: $\tilde{x}_A$ is forward-noised to diffusion step $k_{\mathrm{PBD}} = 3$ and denoised under the modified constraints, after which collision-free sections of the denoised trajectory replace the corresponding invalid sections of the current trajectory. This process is repeated for at most $R_{\mathrm{PBD}} = 5$ refinement attempts, following the original Kuka evaluation setting.

For MPD, whose standard inference starts from Gaussian noise, we use an adapted warm-start variant. We forward-noise $\tilde{x}_A$ to diffusion step $k_{\mathrm{MPD}} = 10$ using the original MPD diffusion process and then apply its standard cost-guided reverse denoising under the modified obstacle configuration. Start and goal configurations are clamped throughout denoising. MPD-WarmStart therefore uses the same invalidated initial trajectory as the other methods rather than generating a new trajectory from pure Gaussian noise.

The diffusion noise levels and refinement budgets are selected on held-out validation environments and fixed for all test problems. MGT + Continuous EmbOpt is initialized from the token embeddings corresponding to z_A and applies 30 steps of continuous embedding-space optimization. MGMP and MGMP w/o Polish start from the same hard-token sequence z_A and apply GGTS; the conditional MGT is used only to propose replacement tokens during refinement, rather than to regenerate the initial trajectory. Full MGMP additionally applies the final continuous polishing stage.

Thus, every method is conditioned on the same invalidated trajectory under identical modified constraints, and no method is allowed to generate a new trajectory from scratch. This construction isolates trajectory-repair capability from differences in initial-generation quality. The average per-step ℓ_2 reconstruction error between x_A and $\tilde{x}_A$ is approximately 1×10^{-4}, and reconstruction does not alter the feasibility of any retained Route A before obstacle relocation, making tokenizer-induced differences negligible in this comparison.

For each successful repair, we compare the repaired trajectory with the shared invalidated initial trajectory in joint space. At each timestep, we compute the ℓ_2 distance between the corresponding joint configurations. Average deviation is the mean of these distances over the trajectory, while maximum deviation is the largest distance at any timestep. A waypoint is considered changed when its joint-space deviation exceeds $\epsilon = 0.1$ rad. Changed span is the longest contiguous sequence of changed waypoints, reported as a percentage of the full trajectory length. Larger deviation values indicate greater changes in joint configuration, while a larger changed span indicates that the repair modifies a broader portion of the route rather than only the local collision region.

B.8 Implementation Details for Unseen Obstacle Layouts

Maze2D. For Maze2D, the VQ-VAE tokenizer uses a 1D ResNet encoder with channel width 16 and depth 4. It applies a temporal downsampling factor of 4 and uses a codebook of 8192 entries with code dimension 16. The tokenizer is trained with a smooth-ℓ_1 reconstruction loss, EMA-reset vector quantization, commitment weight 0.02, and an additional velocity reconstruction loss with weight 0.5. We train the VQ-VAE for 800k iterations with batch size 256 using AdamW and an initial learning rate of 3×10^{-4}, decayed by 0.1 at 250k, 400k, and 600k iterations. Since the trajectory horizon is $H = 48$, each trajectory is compressed into 12 discrete tokens.

The masked generative transformer contains 8 self-attention layers, embedding dimension 256, 16 attention heads, FFN multiplier 4, and dropout 0.2. The six rectangular obstacles are represented by

their 2D center coordinates and encoded using a ViT-1D obstacle encoder. During training, we apply obstacle-conditioning dropout with probability 0.2. The transformer is trained for 200k iterations with batch size 256 using AdamW, learning rate 1×10^{-4}, and weight decay 1×10^{-6}.

At inference time, the refinement objective is $J = w_{\mathrm{SDF}} L_{\mathrm{SDF}} + w_{\mathrm{sg}} L_{\mathrm{sg}}$. Before SDF convergence, defined as $\mathcal{L}_{\mathrm{sdf}} < 5\times10^{-4}$, we use $(w_{\mathrm{sdf}}, w_{\mathrm{sg}}) = (200, 100)$; after convergence, we switch to $(400, 100)$.

7-DoF Kuka and 14-DoF DualKuka. For the Kuka environments, the VQ-VAE uses a 1D ResNet encoder with width 256 and depth 4, together with a codebook of 8192 entries and code dimension 256. The 7-DoF Kuka uses a temporal downsampling factor of 4, producing 12 tokens per trajectory. The 14-DoF DualKuka uses a smaller downsampling factor of 2, producing 24 tokens per trajectory to preserve finer temporal resolution for the higher-dimensional system. In addition to the smooth-ℓ_1 reconstruction and velocity losses, we include a forward-kinematics Cartesian position loss with weight 0.6, computed every 10 steps, to improve the spatial accuracy of reconstructed joint-space trajectories. The VQ-VAE is trained for 800k iterations with batch size 256 using AdamW and an initial learning rate of 3×10^{-4} (1×10^{-4} for Dualkuka), decayed by 0.1 at 250k, 400k, and 600k iterations, with no weight decay (weight decay for Dualkuka is 1×10^{-6}).

The masked generative transformer uses 8 self-attention layers, embedding dimension 512, 16 attention heads, FFN multiplier 4, and dropout 0.2. For 7-DoF Kuka, four box obstacles are represented by their 3D center coordinates and encoded using a ViT-1D obstacle encoder. For 14-DoF DualKuka, five box obstacles are encoded in the same way. During training, obstacle conditioning is randomly dropped with probability 0.2. The transformer is trained for 200k iterations with batch size 256 using AdamW, learning rate 1×10^{-4} and weight decay 1×10^{-4}.

At inference time, MGMP performs two token refinement steps, followed by 5 final latent-embedding optimization steps using momentum $\beta = 0.9$. The SDF-guided refinement weights are set to $(w_{\mathrm{sdf}}, w_{\mathrm{sg}}) = (300, 150)$ before convergence and switched to $(600, 450)$ after SDF convergence.

B.9 Generalization to Additional-Obstacle Environments Setup

Construction of Additional-Obstacle Environments. We construct additional-obstacle evaluation environments by augmenting held-out base scenes rather than sampling new layouts from scratch. The original obstacles are inherited from the corresponding base evaluation environment, and additional obstacles are inserted to increase the constraint density. Additional obstacles are placed using rejection sampling under non-overlap constraints with the inherited obstacles and invalid workspace regions. This avoids degenerate or physically invalid layouts that could arise from independently sampling all obstacle components, such as overlapping obstacles or newly added obstacles occluding predefined free-space regions. The resulting environments are used only for evaluation; no training trajectories are collected for the combined obstacle configurations. Each start-goal problem is pre-verified with BIT* under a 60-second time budget, ensuring that all reported failures correspond to planner limitations rather than infeasible planning instances.

B.10 Generalization to Unseen Obstacle Geometries Environments Setup

To evaluate generalization to unseen obstacle geometries, we replace the box obstacles used during training with object-level meshes at inference time. The mesh library contains ten ShapeNet objects: car, trash bin, hat, open kettle, keyboard, birdcage, guitar, camera, handbag, and closed kettle, as shown in Fig. 6. These objects introduce irregular obstacle boundaries and non-cuboid geometries that are not observed during training. This setting therefore tests whether the planner can generalize to more realistic object-shaped obstacles without additional training.

Each object is represented by a triangulated mesh normalized to fit within a unit cube and then scaled in simulation using an object-specific scalar. The scale is chosen such that the resulting obstacle is larger than the box obstacles used during training. Specifically, the target world-space half-extent is set to approximately 0.34m, corresponding to $1.7\times$ the training box half-extent of 0.20m. Thus, this setting evaluates not only shape generalization, but also robustness to larger obstacle geometries than those observed during training.



Figure 6: Object-level mesh obstacles used for evaluating generalization to unseen obstacle geometries. These objects replace the box obstacles used during training, creating test environments with irregular and previously unseen obstacle shapes.

The environment and problem generation protocol follows the additional-obstacle setting. We use the same number of environments and start-goal problems, and sample obstacle center positions from the same workspace placement rules used for the additional-obstacle experiments. The orientation of each mesh object is fixed across environments, while only its center position varies. This ensures that the main distribution shift comes from obstacle geometry and size rather than from a different placement strategy. For 7-DoF Kuka, the four box obstacle slots are replaced by four mesh objects. For 14-DoF DualKuka, we additionally include the fifth mesh obstacle. All start and goal configurations are generated using the same feasibility procedure as in the additional-obstacle experiments.

For SDF-guided refinement, we use DeepSDF to provide a differentiable signed-distance representation for each mesh object. Each object is associated with a fixed latent code $z \in \mathbb{R}^{64}$ from a precomputed latent-code library. The library contains ten latent codes for the full ShapeNet object set, from which four codes are selected per Kuka scene and five per DualKuka scene. The latent codes are fixed during planning; we do not optimize the object latent code at test time.

Given a query point x_w in world coordinates and an obstacle centered at c, we first transform the point into the canonical coordinate frame used by DeepSDF:

$$x_{\text{obj}} = R\left(\frac{x_w - c}{s} - m\right) + t, \tag{6}$$

where s is the object scale, $m \in \mathbb{R}^3$ is the AABB centroid of the normalized mesh, and (R, t) denotes the fixed rigid alignment between the normalized mesh frame and the DeepSDF training frame. The AABB-centroid correction accounts for meshes whose normalized geometry is not centered at the origin. The world-space signed distance is then recovered by scaling the canonical DeepSDF output:

$$d(x_w) = s \cdot f_\theta(z, x_{\text{obj}}), \tag{7}$$

where f_θ is the DeepSDF decoder. Negative values indicate that the query point lies inside the object, while positive values indicate free space.

During planning, DeepSDF is used to compute differentiable collision losses for SDF-guided code refinement. Final trajectory validity is evaluated separately in PyBullet. This separation allows the planner to use smooth SDF gradients for refinement while retaining a physics-engine-based collision check for evaluation.

Table 9: Effect of refinement budget on Ring Maze. SR and VSF denote success rate and valid-sample fraction, respectively.

(a) CHOMP: optimization iterations

Iterations	10	20	30	50
SR (%) ↑	73.30	73.30	73.30	73.30
VSF (%) ↑	22.30	23.30	23.70	23.30

(b) Learned refinement: number of guidance steps

Steps	0	1	2	3	4	5
PBD SR (%) ↑	40.95	53.63	59.30	63.05	65.55	67.37
PBD VSF (%) ↑	7.80	12.67	15.82	18.14	21.95	26.00
MPD SR (%) ↑	2.00	7.80	11.60	**14.40**	10.30	9.30
MPD VSF (%) ↑	0.27	1.00	1.62	**2.25**	1.85	1.80

C Additional Simulation Results

This section provides qualitative results for the simulated planning tasks. Each figure compares representative trajectories generated by BIT*, PBD, and MGMP. These visualizations complement the quantitative results in the main paper by showing the geometric quality of the planned trajectories.

C.1 Further Analysis on Ring Maze

We provide additional analyses to better understand when local refinement succeeds or fails in the Ring Maze benchmark.

Effect of refinement budget. Table 9 studies whether the performance gap can be closed simply by increasing the amount of local refinement. For CHOMP, increasing the number of optimization iterations from 10 to 50 has almost no effect: the success rate remains at 73.3%, while the valid-sample fraction varies only between 22.3% and 23.7%. Increasing the refinement budget therefore does not substantially change the solutions reached by the optimizer. For learned guidance, most of the improvement occurs within the first three refinement steps. PBD improves substantially during the first three steps, but the gains become small thereafter: from step 3 to step 5, success increases only from 63.05% to 67.37%. In contrast, MPD peaks after three guidance steps (14.40% success) and degrades with further refinement. These results indicate that simply increasing the continuous-refinement budget does not reliably overcome the planning basins encountered in Ring Maze.

Effect of collision severity. We further compare PBD and MGMP as the initial trajectory moves farther from feasibility. We measure collision severity using the minimum clearance of the initial trajectory to the obstacles, where negative values indicate penetration. We group trajectories into three ranges: $[-0.05, 0)$, $[-0.15, -0.05)$, and $[-0.30, -0.15)$. PBD repairs 46.5%, 9.6%, and 5.2% of trajectories in these three ranges, respectively, whereas MGMP achieves 93.8%, 84.1%, and 71.4%. The widening gap with deeper collisions suggests that gradient-based local refinement is effective near the feasible region, whereas GGTS remains more effective when larger trajectory modifications are required.

C.2 Generalization to Unseen Obstacle Layouts

Figure 7 shows qualitative results on the Maze2D environments. Across both Maze2D-6 and Concave Maze2D, MGMP generates collision-free paths that reach the goal while avoiding unseen obstacle layouts. Figures 8–11 further show representative Kuka and DualKuka trajectories. MGMP produces feasible joint-space plans in cluttered scenes.

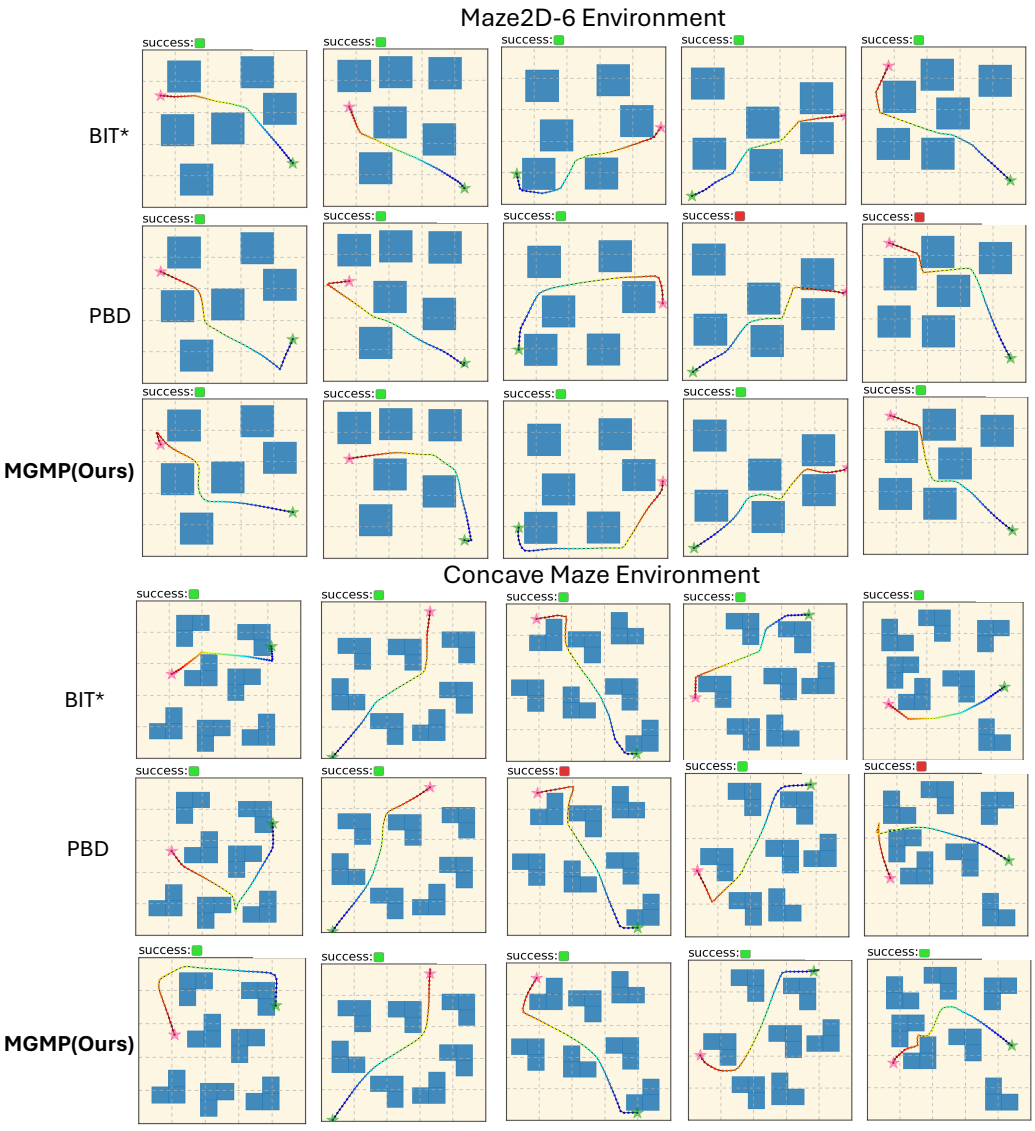


Figure 7: Qualitative results on unseen obstacle layouts in Maze2D. We compare BIT*, PBD, and MGMP on Maze2D-6 and Concave Maze2D. MGMP generates collision-free trajectories that reach the goal while avoiding unseen obstacle placements.

C.3 Generalization to Additional-Obstacle Environments

Figures 12 and 13 show qualitative results in additional-obstacle DualKuka environments. These scenes contain more obstacles than those used during training, making collision-free planning more constrained. MGMP still generates feasible trajectories by combining obstacle-conditioned predictions and applying collision-aware refinement.

C.4 Generalization to Unseen Obstacle Geometries

Figures 14 and 15 show qualitative results when box training obstacles are replaced by unseen object-level mesh obstacles. These examples illustrate the ability of MGMP to adapt to irregular obstacle shapes using SDF-guided code refinement. In contrast to the axis-aligned cuboid settings, these environments require the planner to avoid more realistic and non-convex object geometries.

Table 10: Refinement configurations used in the ablation study. For GGTS variants, discrete refinement consists of position selection, candidate shortlisting, exact verification, and token commitment, followed by optional continuous polishing.

Variant	Position Selection	Candidate Shortlist	Exact Verification	Commit	Continuous Opt.
MGT (Conf.-Refine)	Confidence	MGT prob.	–	–	–
MGT + EmbOpt-30	–	–	–	–	EmbOpt-30
Conf.-only Remask	Confidence	Eq. (2), $M \to k$	top-k	Sequential	Polish
Geom.-only Remask	Geometry	Eq. (2), $M \to k$	top-k	Sequential	Polish
Probability-only Screening	Geom.+Conf.	MGT prob., $M \to k$	top-k	Sequential	Polish
Independent Exact Search	Geom.+Conf.	Eq. (2), $M \to k$	top-k	Independent	Polish
MGMP w/o Polish	Geom.+Conf.	Eq. (2), $M \to k$	top-k	Sequential	–
Full MGMP	**Geom.+Conf.**	**Eq. (2),** $M \to k$	**top-**k	**Sequential**	**Polish**

For all GGTS variants, token positions associated with detected collision timesteps are additionally included in the remasking set. Unless otherwise stated, all variants use the same proposal size M, shortlist size k, refinement rounds, and polishing budget.

Table 11: Additional analysis of GGTS on 7-DoF Kuka. (a–c) Sensitivity to the candidate shortlist size, refinement schedule, and continuous polishing budget. The MGT proposal size is fixed to $M = 16$, and k denotes the number of candidates retained for exact verification. For the refinement schedule, 1×4, 2×2, and 4×1 use the same nominal refinement budget, while 1×1 serves as a minimal-refinement reference. (d) Comparison of first-order screening and exact verification. Default settings are shown in bold.

(a) Candidate shortlist size k

k	SR ↑	VSF ↑	RT ↓
1	0.95	0.84	0.38
2	0.97	0.87	0.44
4	0.97	0.88	0.56
16	0.98	0.89	1.28

(b) Refinement schedule

Schedule	SR ↑	VSF ↑	RT ↓
1×1	0.93	0.79	0.34
1×4	0.95	0.83	0.40
$\mathbf{2 \times 2}$	0.97	0.87	0.44
4×1	0.96	0.85	0.52

(c) Continuous polishing

Steps	SR ↑	VSF ↑	RT ↓
0	0.94	0.83	0.40
3	0.96	0.86	0.42
5	0.97	0.87	0.44
10	0.97	0.88	0.48

(d) Screening and exact verification

Variant	SR ↑	VSF ↑	Exact Evals. ↓	RT ↓
First-order Only	0.93	0.78	0	0.32
Full GGTS	0.97	0.87	8	0.44
Exhaustive Verification	0.98	0.89	64	1.28

C.5 Ablation Study And Hyperparameter Experiments

Screening and Exact Verification We further analyze the two-stage candidate evaluation used in GGTS. First-order Only directly commits the top-ranked candidate without exact verification, while Exhaustive Verification evaluates all $M = 16$ proposals using the exact objective. As shown in Table 11(d), First-order Only reduces SR/VSF to $0.93/0.78$, indicating that the first-order score is effective for screening but insufficient as the final discrete decision rule. Exhaustive Verification achieves $0.98/0.89$ SR/VSF, close to Full GGTS at $0.97/0.87$, but requires $8\times$ more exact evaluations and increases runtime from 0.44 s to 1.28 s. Thus, the proposed $M \to k$ screening retains most of the benefit of exhaustive verification while substantially reducing verification cost.

Refinement Hyperparameters Table 11(a–c) studies the sensitivity of GGTS to the candidate shortlist size, refinement schedule, and continuous polishing budget on 7-DoF Kuka. For candidate screening, performance improves from 0.95 SR / 0.84 VSF at $k = 1$ to $0.97/0.87$ at $k = 2$, while further increasing k to 4 or 16 provides only marginal gains at substantially higher runtime. We therefore use $k = 2$, reducing the $M = 16$ MGT proposals to only two candidates for exact verification.

For the refinement schedule, 2×2 achieves the best overall performance among the equal-budget configurations, reaching 0.97 SR and 0.87 VSF, compared with $0.95/0.83$ for 1×4 and $0.96/0.85$ for 4×1. Finally, continuous polishing improves feasibility from $0.94/0.83$ SR/VSF without polishing

to $0.97/0.87$ after five steps, while increasing the budget to ten steps yields only a marginal VSF improvement to 0.88 with no further gain in SR. We therefore use $k = 2$, a 2×2 refinement schedule, and five polishing steps in the main experiments.

D Real-World Experiments

D.1 Environment setup

We conduct real-world tabletop obstacle-avoidance experiments using the Baxter robot's left arm. Figure 16 shows the real-world setups used for data collection. Figures 16(a)–(c) show the 3D trajectory setting, where box obstacles are placed on the table and the robot moves through the workspace while avoiding them. This setting is used for shelf placing and button pressing. Figure 16(d) shows the 2D tabletop drawing setting, where the robot holds a pen and follows a near-surface trajectory on the table.

Training. We collect two real-world datasets of task-agnostic trajectories. The 3D dataset contains 200 trajectories across 20 obstacle configurations, with 10 trajectories per configuration. Demonstrations are collected by guiding the arm in zero-gravity mode from a start region to a goal region while avoiding four tabletop box obstacles. Obstacle centers are measured using HTC Vive trackers, and robot joint angles are recorded along each trajectory. The 2D tabletop dataset contains 200 near-surface gripper trajectories across 10 obstacle configurations for pen drawing.

Evaluation on downstream tasks. For shelf placing and button pressing, MGMP generates a trajectory to a task-specific goal; the robot follows this trajectory then executes a simple hand-coded primitive action, i.e. releasing the object on the shelf or moving the end-effector downward to press the button. For drawing, the robot grasps a pen and follows the 2D trajectory produced by MGMP to draw a line.

D.2 Qualitative results

Figures 17–19 show qualitative results from the real-world experiments. Figure 17 presents the pen-drawing task. In successful cases, the robot follows the planned tabletop trajectory and draws a continuous line toward the target region while avoiding surrounding obstacles. Failures occur when the trajectory is blocked by nearby boxes, leading to obstacle contact or preventing the pen from reaching the desired final location.

Figure 18 shows qualitative results for button pressing. In successful trials, MGMP plans a collision-free approach trajectory that brings the end-effector above the button, after which the downward primitive completes the press. The failure case occurs when the generated trajectory does not place the end-effector at a suitable pre-press pose; as a result, the robot fails to reach the task-specific goal required for activating the button.

Figure 19 shows qualitative results for shelf placing under three modes with different target shelf regions. In successful trials, MGMP generates trajectories that move the object around obstacles and reach valid release poses near the target shelf region. Failures occur when the trajectory collides with a nearby obstacle, as shown in row (c), or when the planned trajectory approaches the shelf from an unsuitable direction and is blocked by the upper shelf layer, as shown in row (g). Rows (b) and (c) use the same environment but different planned trajectories for placing the cube, illustrating the multimodality of MGMP's generated actions.

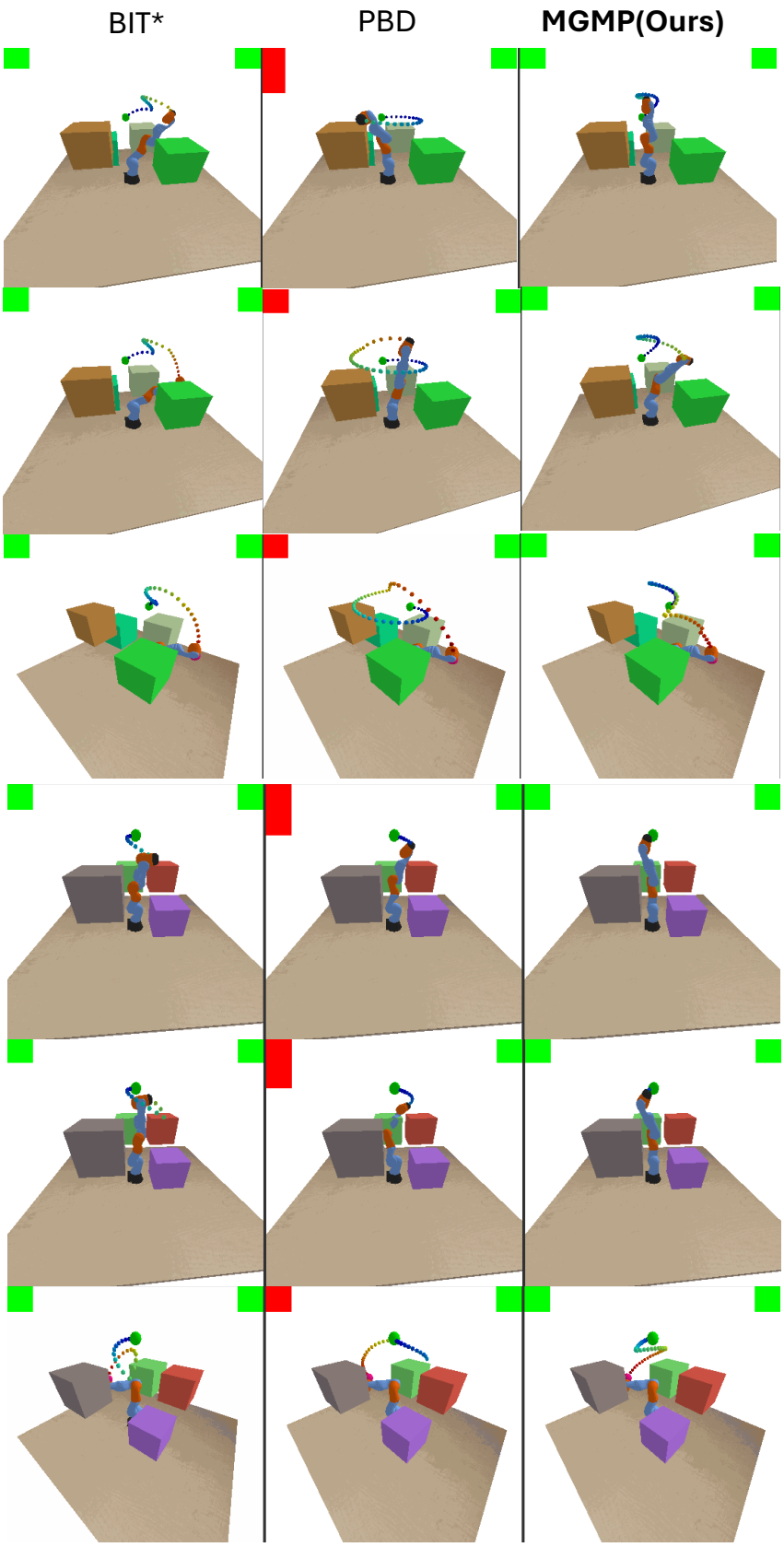


Figure 8: Qualitative results on unseen obstacle layouts in 7-DoF Kuka. Each column compares trajectories generated by BIT*, PBD, and MGMP under the same obstacle configuration. MGMP generates feasible joint-space trajectories that avoid cluttered obstacles.

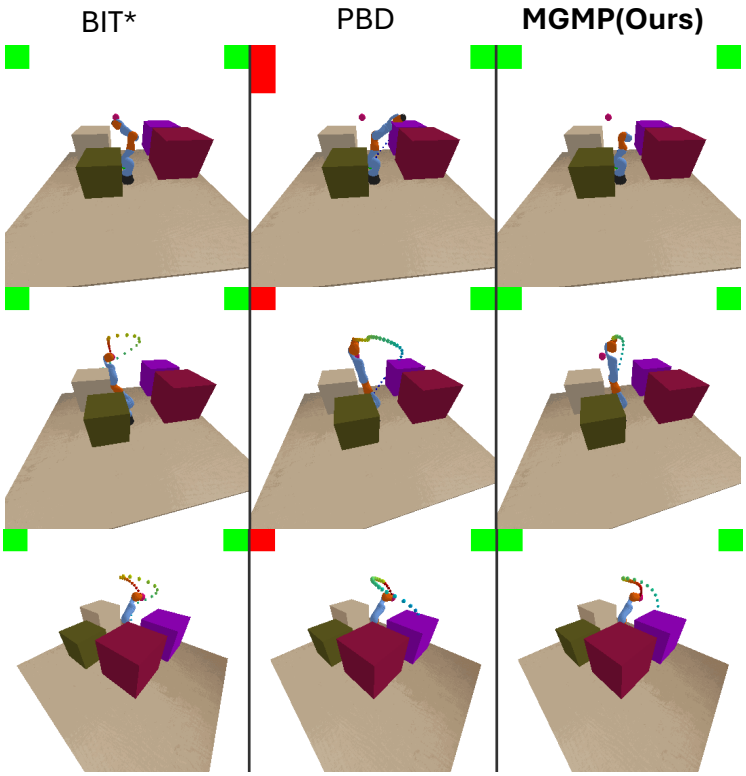


Figure 9: Additional qualitative results on unseen obstacle layouts in 7-DoF Kuka. MGMP produces collision-free plans across different start-goal pairs and obstacle placements.

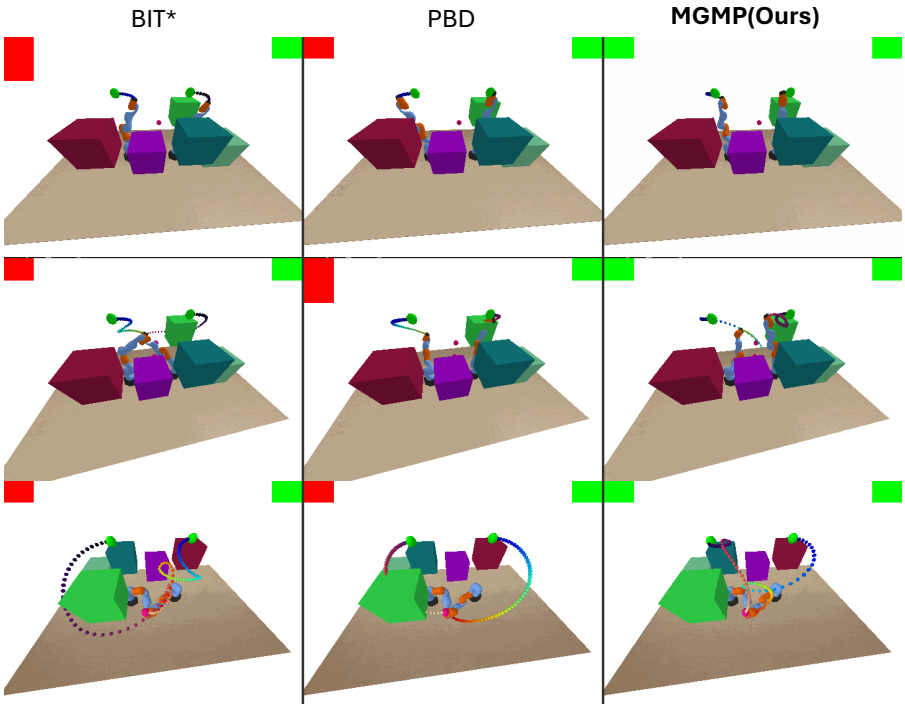


Figure 10: Qualitative results on unseen obstacle layouts in 14-DoF DualKuka. The examples compare BIT*, PBD, and MGMP in higher-dimensional dual-arm planning scenes. MGMP generates feasible trajectories while avoiding both workspace obstacles and inter-arm collisions.

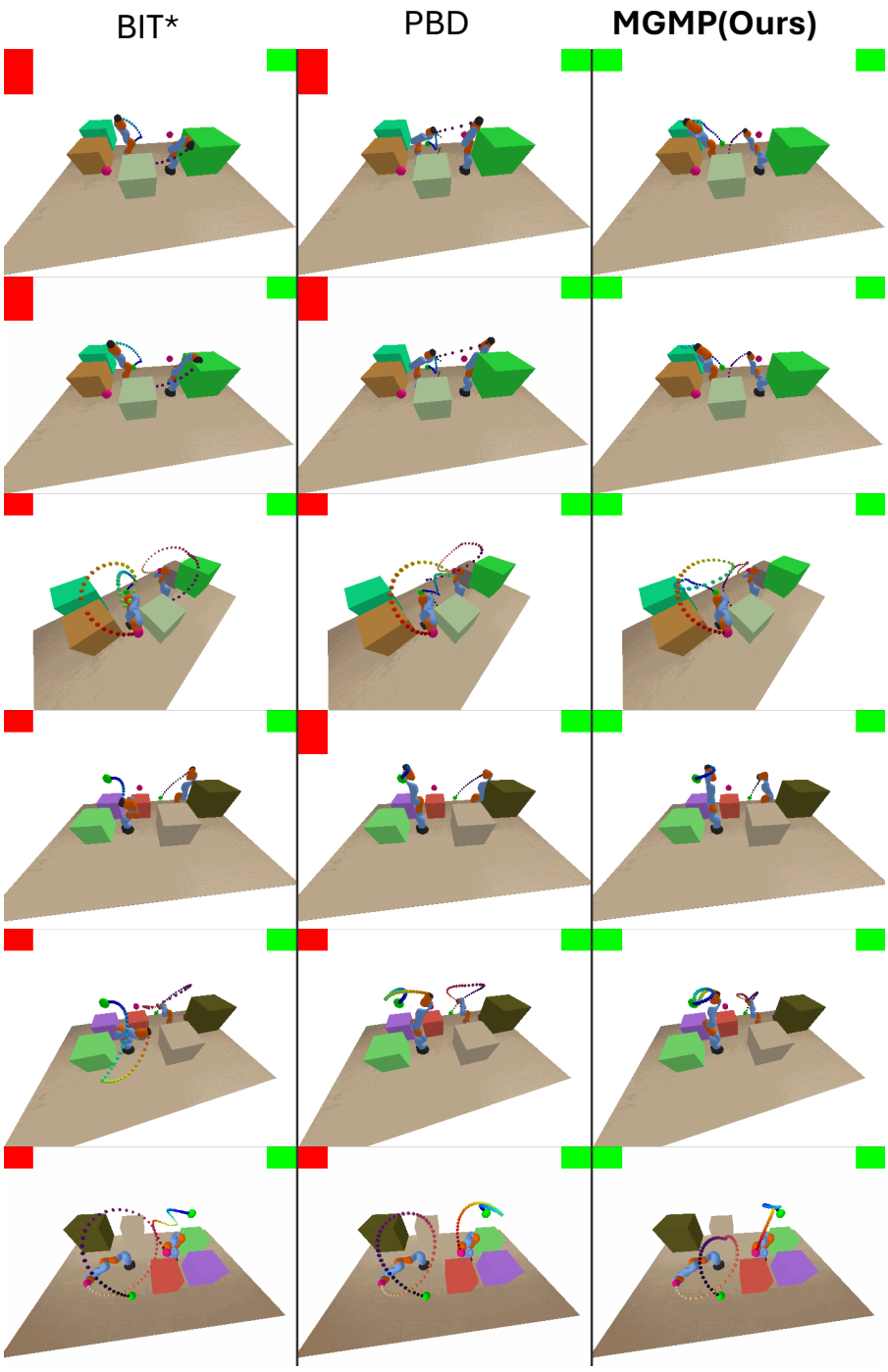


Figure 11: Additional qualitative results on unseen obstacle layouts in 14-DoF DualKuka. MGMP remains robust across different obstacle placements and start-goal configurations.

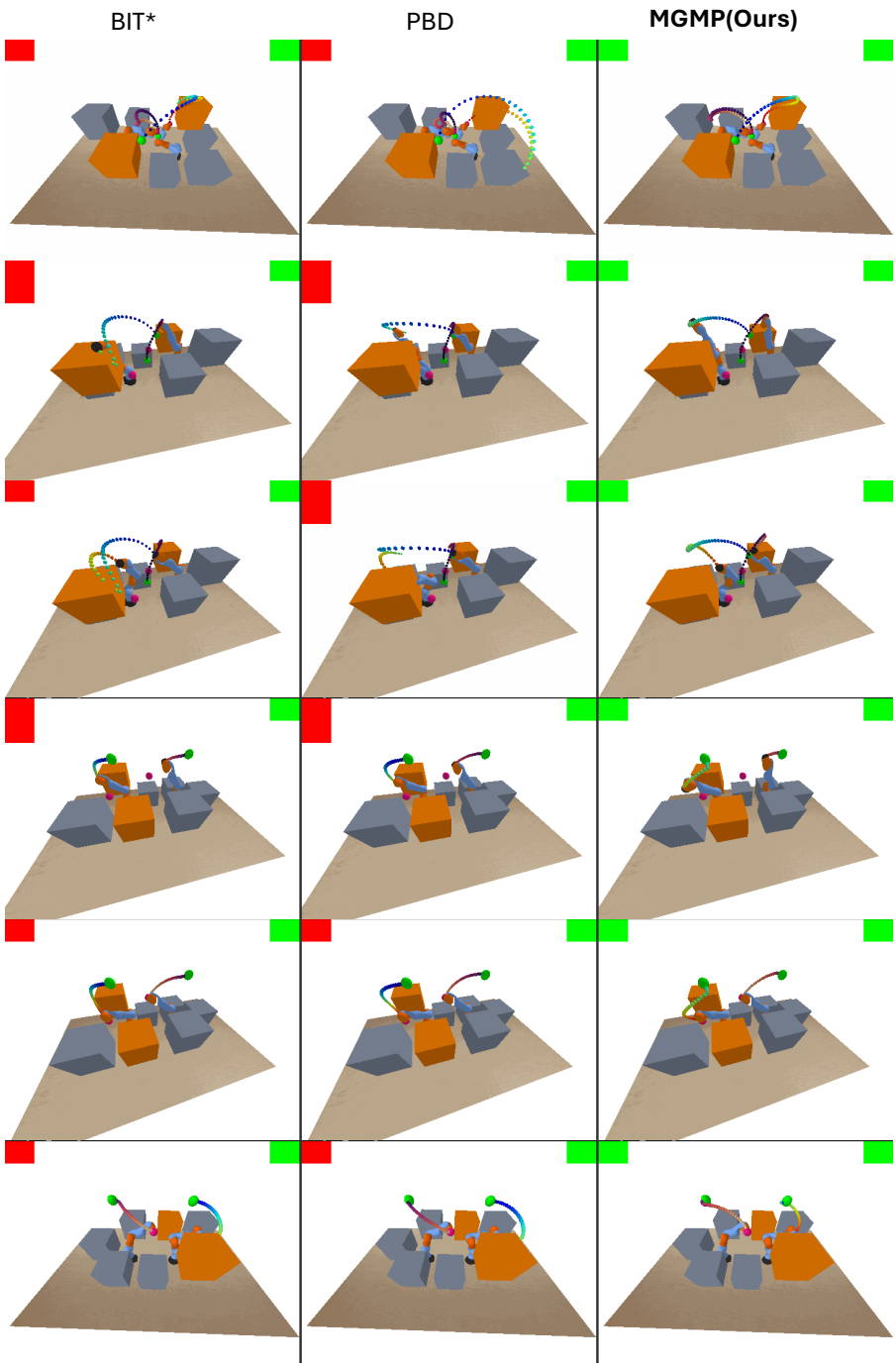


Figure 12: Qualitative results on additional-obstacle 14-DoF DualKuka environments. Compared with the base setting, these scenes include more obstacles at test time. MGMP generates collision-free trajectories in the more constrained workspace.

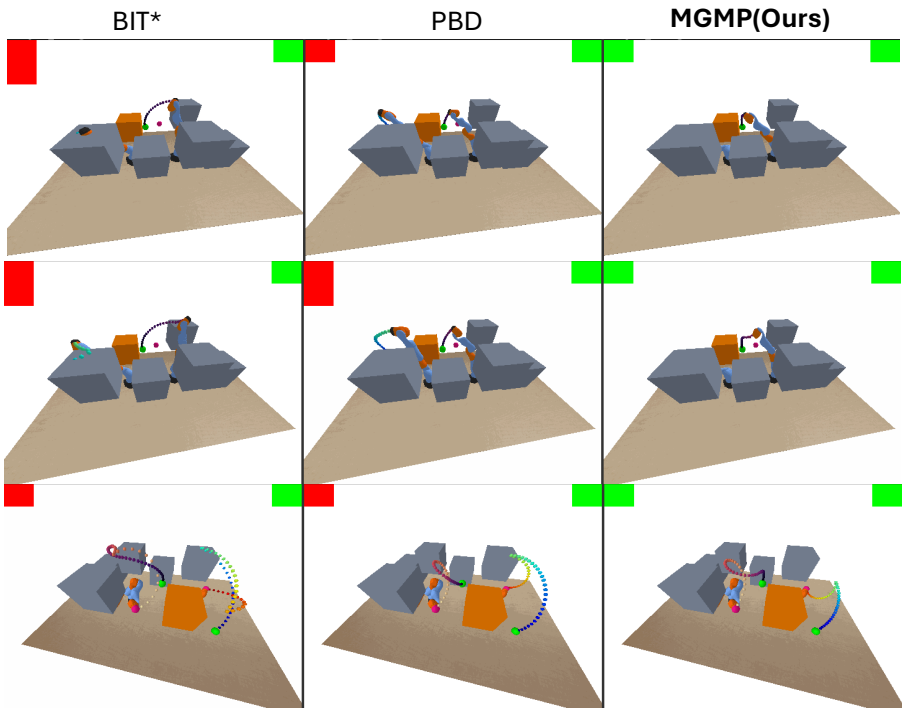


Figure 13: Additional qualitative results on additional-obstacle 14-DoF DualKuka environments. MGMP remains effective when the test scene contains more constraints than those observed during training.

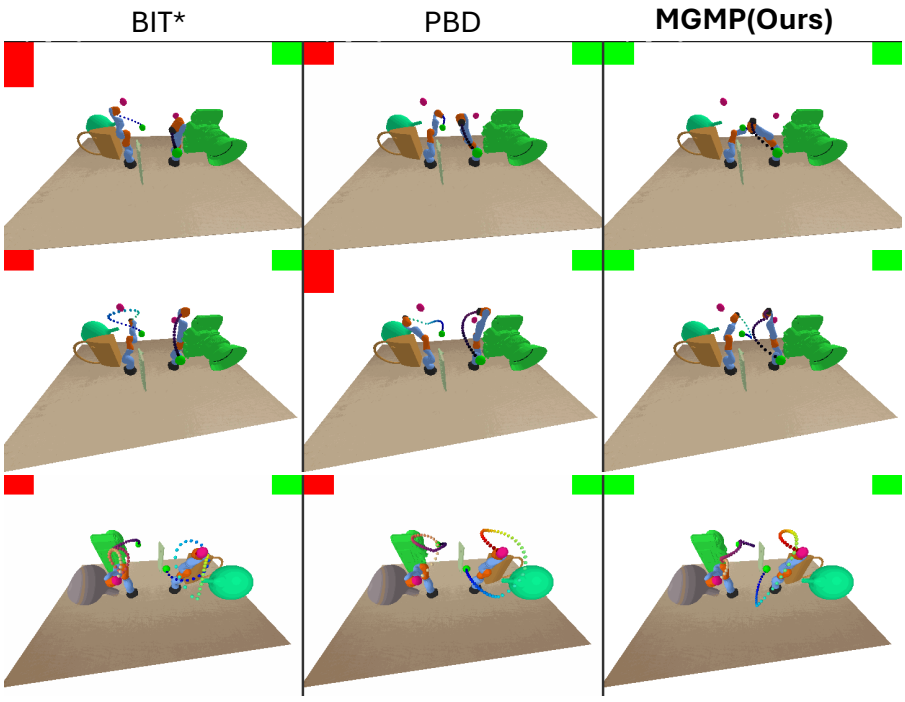


Figure 14: Qualitative results on unseen obstacle geometries in 14-DoF DualKuka. Cuboid training obstacles are replaced by unseen object-level mesh obstacles. MGMP adapts to these geometry shifts and generates collision-free trajectories around irregular objects.

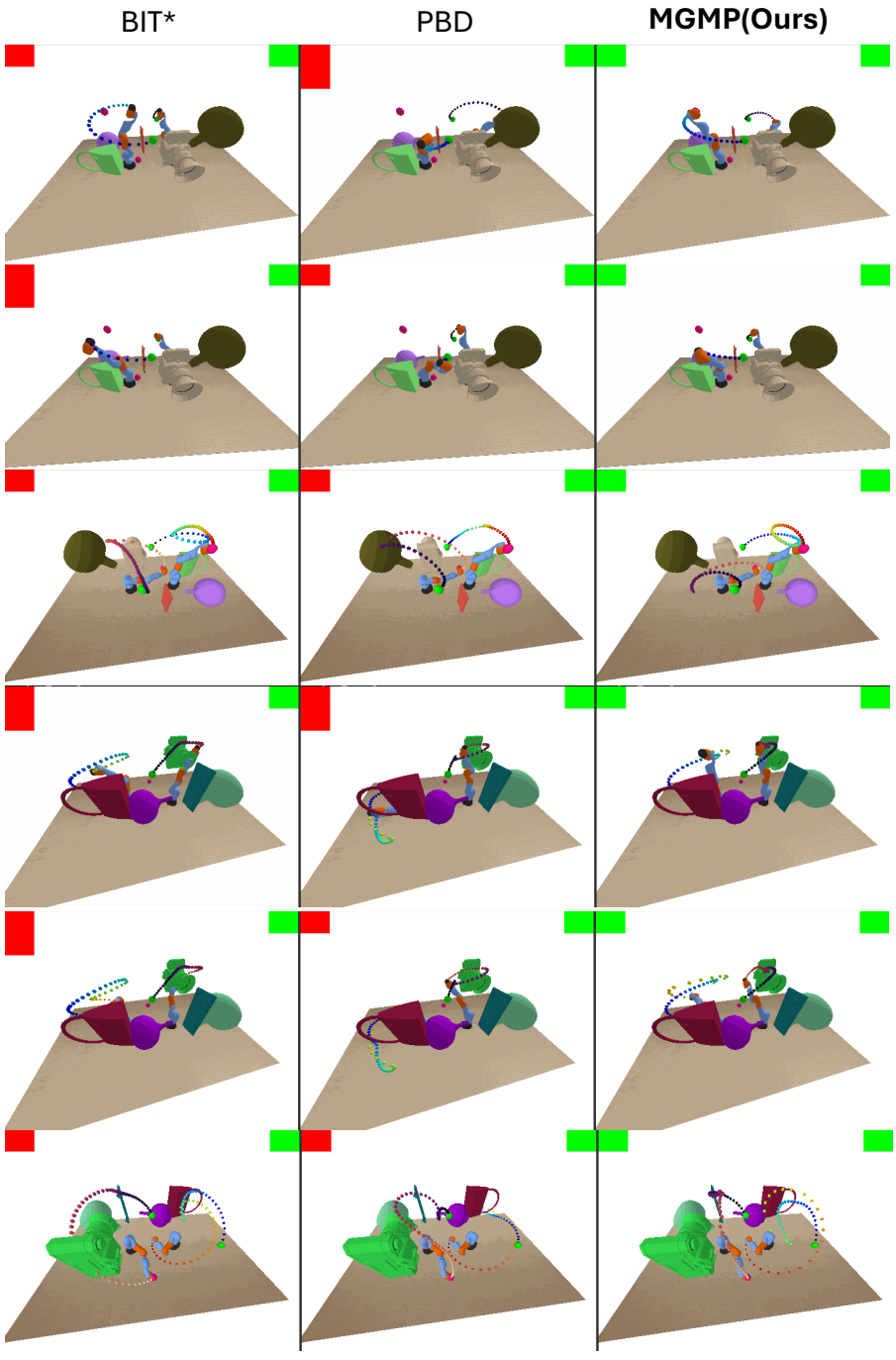


Figure 15: Additional qualitative results on unseen obstacle geometries in 14-DoF DualKuka. MGMP maintains feasible planning performance under realistic mesh obstacles that differ from the box obstacles used during training.



Figure 16: Real-world tabletop setups for Baxter experiments. Figures (a)–(c) show the 3D trajectory setting used for shelf placing and button pressing, where the robot plans obstacle-avoiding motions through the workspace. Panel (d) shows the 2D tabletop drawing setting, where the robot follows a near-surface trajectory while holding a pen.

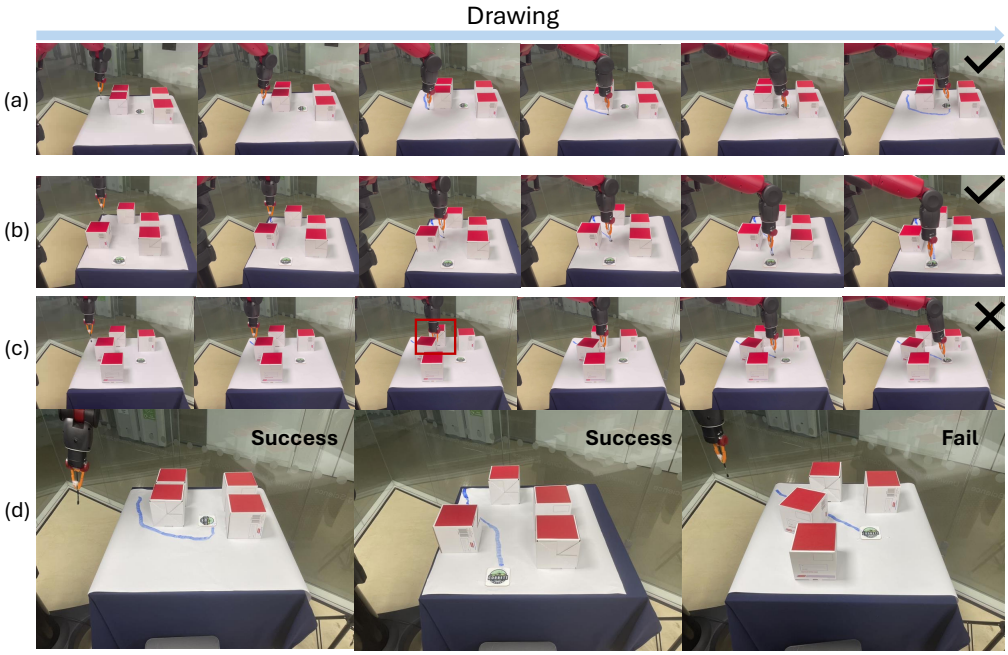


Figure 17: Qualitative results for the real-world pen-drawing task. Rows (a) and (b) show successful executions, where the robot follows the planned 2D tabletop trajectory and draws a line to the target while avoiding obstacles. Row (c) shows a failure case caused by collision with an obstacle during execution. Row (d) shows final drawn trajectories.

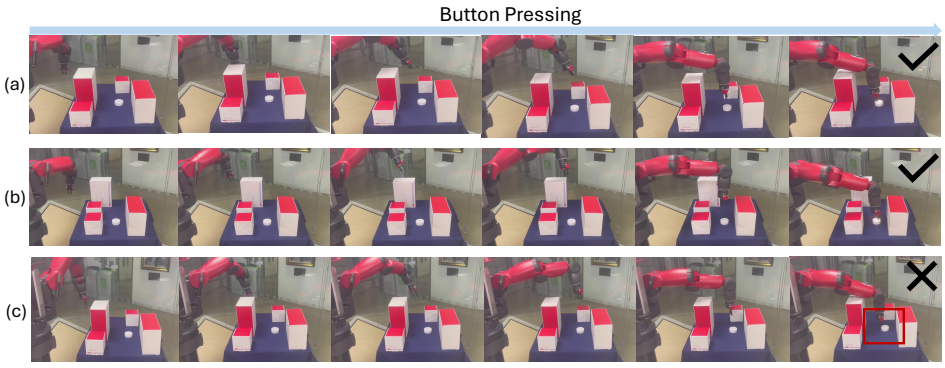


Figure 18: Qualitative results for the real-world button-pressing task. Rows (a) and (b) show successful executions. Row (c) shows a failure case: the robot does not reach the correct pre-press goal pose.

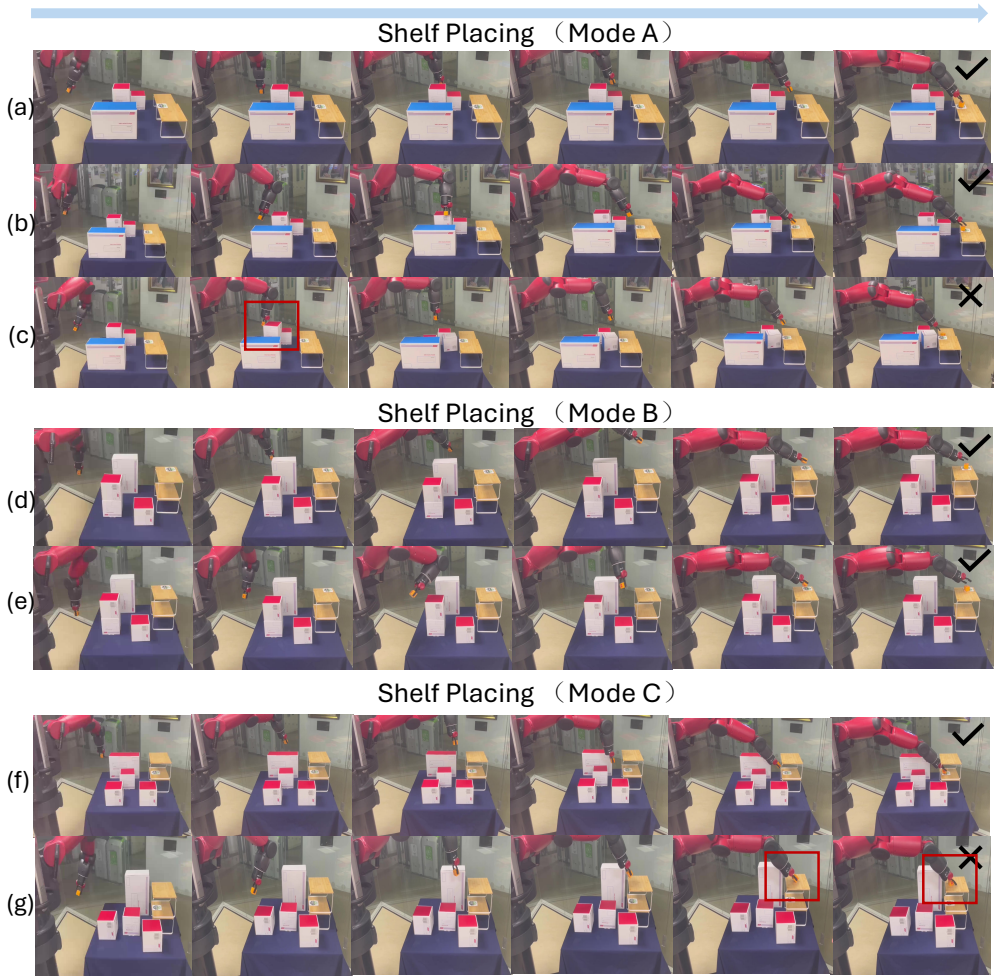


Figure 19: Qualitative results for the real-world shelf-placing task under three target shelf modes. MGMP successfully generates obstacle-avoiding trajectories that reach valid release poses. Failure cases are mainly caused by obstacle collisions or unsuitable approach trajectories that prevent successful placement.