MOSAIC: A Skill Centric Algorithmic Framework for Long-Horizon Manipulation Planning

Paper to Cover

- **Title:** MOSAIC: A Skill-Centric Algorithmic Framework for Long-Horizon Manipulation Planning
- **Authors:** Itamar Mishani, Yorai Shaoul, and Maxim Likhachev
- **Affiliation:** Robotics Institute, School of Computer Science, Carnegie Mellon University
- **Year:** 2025 (arXiv v2: November 2025)
- **Website:** skill-mosaic.github.io
- **ArXiv:** https://arxiv.org/abs/2504.16738

TL/DR Summary

- **Proposes** a multi-directional planning framework that composes generic, imperfect skills (e.g., push, pick, transport) to solve long-horizon manipulation tasks
    - Without requiring hand-coded symbolic task specifications

- **Key Idea** is instead of searching forward from start or backward from goal, MOSAIC discovers regions where skills are likely to succeed ("islands of competence")
    - Connects them using physics simulation to verify feasibility

- **Addresses** the problem of existing methods struggling when solutions require non-obvious intermediate steps that don't show immediate progress toward the goal
    - Additionally, reasons over where skills are likely to succeed

Motivations

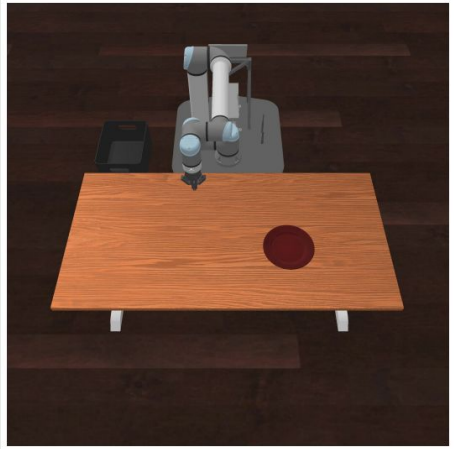
Two mainstream approaches for long-horizon manipulation

- **TAMP:** Strong at long-horizon reasoning, but requires hand-engineered symbolic representations that are brittle
- **Policy Learning:** Learns complex skills, but struggles with long-horizon composition and generalizes poorly
- **Common flaw:** Both rely on goal-directed search, which fails when solutions require non-obvious intermediate steps
- **Gap:** Skills are imperfect — no existing framework reasons about where skills are likely to succeed!

For Example:

- The plate sits deep on the table and cannot be grasped directly due to its geometry
- It must first push the plate to the edge then pick it from the side to transport to the bin

This requires non-obvious intermediate steps, exactly where goal-directed search struggles!



Related Work

Model-Based Planning with Simulation

- Uses physics simulators (MuJoCo, Sapien) in the planning loop to predict action outcomes → MOSAIC builds on this

Task and Motion Planning (TAMP)

- Interleaves symbolic task planning with motion planning
- Powerful but requires hand-engineered symbolic representations

Single Policy Methods

- Diffusion models and RL learn short-horizon skills well, but struggle with long-horizon composition and generalization

Skill-Based Methods

- Compose skills via chaining (options framework)
- All existing methods are goal-directed — none reason about where skills are likely to succeed.

Background to Problem Def.

Options Framework & Skill Chaining:

- **Options Framework** (Sutton et al., 1999): An option is a temporally extended action with an initiation set (where it can start), a policy, and a termination condition
- **Skill Chaining** (Konidaris & Barto, 2009): Composes options sequentially → each new skill must start where the previous one ended

Problem:

- Exploration is constrained to local neighborhoods around the current chain, making it hard to discover useful but distant intermediate configurations
- Requires explicit initiation/termination/effect sets for each skill

Problem Definition

Skills, Generators, Connectors & BVPs:

Building on the options framework,a skill σ is defined as $(\pi_\sigma, \Theta_\sigma)$ — a policy and its parameter space. Where Parameters are the inputs that control how the skill executes (e.g., where to push an object, which grasp to attempt), producing a trajectory τ

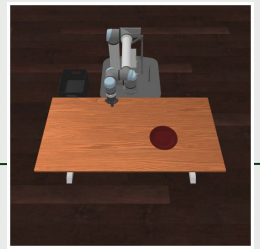
Two types are distinguished:

- **Generators G**: $\Theta \rightarrow T$
  - Produce trajectories freely, without requiring start/goal states
  - Answers: "**Where can I do something useful?**"
- **Connectors C**: $\Xi \times \Xi \times \Theta \rightarrow T$
  - Produce trajectories conditioned on start and goal conditions
  - Solve boundary value problems: "**Can I link trajectory A's end to trajectory B's start?**"

A Boundary Value Problem in this context means:

- Given a desired start and end condition, find a trajectory connecting them

Together they define the skill space: A = U Generators U Connectors



A Push generator freely pushes the plate to a new position — creating an island.

A Pick generator attempts a grasp at the table edge — creating another island.

A Push connector then bridges the gap — moving the plate and positioning the robot from where the first trajectory ended to where the second begins

Problem Definition

Planning Objective & Underactuation:

Given start state x_{start} and goal condition ξ_{goal}, find a sequence of N skills producing trajectories $\{\tau_1, \ldots, \tau_N\}$ satisfying:

- $\tau_1(0) = x_{start}$ — starts at start state
- $\tau_i(1) = \tau_{\{i+1\}}(0)$ — adjacent trajectories connect continuously
- $\xi_{goal}(\tau_N(1)) = 1$ — final state satisfies the goal

Why is this hard for manipulation?

- The system is underactuated: state space $X = QR \times Q_{O1} \times \ldots \times Q_{Ok}$ includes both robot and object configurations
- Objects aren't directly actuated → a physics simulator is needed to predict how robot actions affect them

Assumptions

- Access to a high-fidelity physics simulator that can evaluate skill execution outcomes
- A predefined library of parameterized skills is provided (not learned by MOSAIC)
- Skills produce only valid trajectories: collision-free with static obstacles and respecting robot dynamics
- Goal is specified as a binary condition function $\xi_{goal}: X \rightarrow \{0,1\}$
- For theoretical guarantee proofs(probabilistic completeness): deterministic skill-trajectory generation is assumed
    - (i.e., each skill-parameter pair always produces the same trajectory or batch of trajectories)

Methodology

The Mosaic Graph

MOSAIC constructs a directed multigraph (a graph allowing multiple edges between node pairs), called the mosaic graph:

- Nodes = (skill, parameters, trajectory) — created by Generators
- Edges = (skill, parameters, boundary conditions) — created by Connectors

Generators discover disconnected "islands of competence." Connectors link them by solving boundary value problems. An oracle module guides which skills to invoke and where to connect.

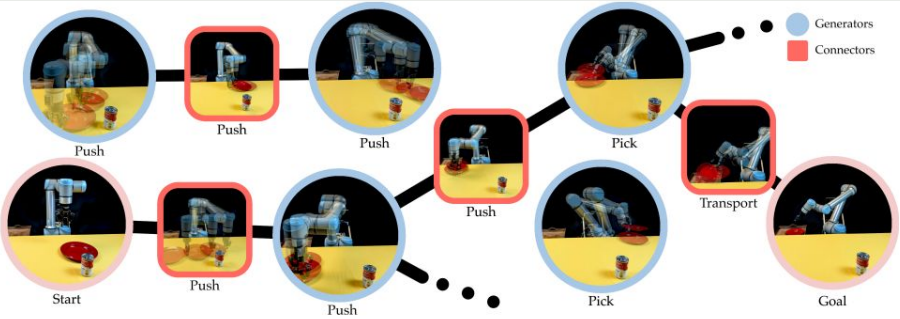


Fig. 1: MOSAIC solves long-horizon manipulation tasks by generating local skill trajectories (circles) and connecting those with connector skills (squares). MOSAIC capitalizes on the skills themselves to guide the exploration process toward regions where they are likely to succeed – enabling effective composition of generic local skills to solve complex tasks.

Methodology

Algorithm 1: MOSAIC

Input: Start state $x_{\text{start}} \in \mathcal{X}$
Goal termination condition function $\xi_{goal} : \mathcal{X} \to \{0, 1\}$
Skill library $\Sigma = \{\sigma\}_{m=1}^{M}$
Oracle O for skill and trajectory selection
Output: Sequence of skills and their parameters Π

1 $\mathcal{M}$ = DirectedMultigraph()
2 $\mathcal{G} \leftarrow \{\mathcal{G}_i \in \Sigma\}$ // Generators container
3 $\mathcal{C} \leftarrow \{\mathcal{C}_j \in \Sigma\}$ // Connectors container
4 **while** $\mathcal{M}$.NODES $= \emptyset$ **do**
5 **for** $\sigma \in \mathcal{G}$ **do**
6 $\theta \leftarrow O$.SampleParameters(σ)
7 $\tau \leftarrow \sigma(\theta)$ // Estimatd, valid trajectories
8 **if** $\tau \neq \emptyset$ **then**
9 $\mathcal{M}$.AddNode$((\sigma, \theta, \tau)$, Cost$(\tau))$
10 **end**
11 **end**
12 **end**
13 $\tau_{\text{start}} \leftarrow \{x_{\text{start}}\}$
14 $\mathcal{M}$.AddNode$((\emptyset, \emptyset, \tau_{\text{start}}), 0)$; // Add start state as a node, with zero cost
15 **while** $\neg\mathcal{M}$.HasPath$(x_{start}, \{x \mid \xi_{goal}(x) = 1\})$ **do**
  // Oracle selects next skill to apply
16 $\sigma \leftarrow O$.ChooseSkill$(\Sigma, \mathcal{M})$
17 $\theta \leftarrow O$.SampleParameters(σ)
18 **if** $\sigma \in \mathcal{C}$ **then**
  // Apply connector skill
19 $\xi_0, \xi_1 \leftarrow O$.ChooseCondsToConnect$(\mathcal{M})$
20 $\tau \leftarrow \sigma(\xi_0, \xi_1, \theta)$
21 **if** $\tau \neq \emptyset$ **then**
22 $\mathcal{M}$.AddEdge$((\sigma, \xi_0, \xi_1, \theta)$, Cost$(\tau))$
23 **end**
24 **else**
  // Apply generator skill
25 $\tau \leftarrow \sigma(\theta)$
26 **if** $\tau \neq \emptyset$ **then**
27 $\mathcal{M}$.AddNode$((\sigma, \theta, \tau)$, Cost$(\tau))$
28 **end**
29 **end**
30 **end**
  // Least-cost path to any goal state
31 **return** $\mathcal{M}$.ShortestPath$(x_{\text{start}}, \{x \mid \xi_{goal}(x) = 1\})$

Main Algorithm

Initialization:

- All generators are invoked with randomly sampled parameters (lines 4-12)
- Each produces a trajectory, a full sequence of robot actions and resulting object states (e.g., a push moving the plate to a new position)
- Each trajectory is rolled out in Sapien to verify physical feasibility
- Valid trajectories become disconnected nodes (islands of competence) in the mosaic graph
- Invalid trajectories are discarded
- If no generators produce valid trajectories, the process repeats with new parameters until at least one node exists
- x_start is added as a zero-cost node (line 14)

Methodology

Algorithm 1: MOSAIC

Input: Start state $x_{\text{start}} \in \mathcal{X}$
Goal termination condition function $\xi_{goal} : \mathcal{X} \to \{0, 1\}$
Skill library $\Sigma = \{\sigma\}_{m=1}^{M}$
Oracle O for skill and trajectory selection
Output: Sequence of skills and their parameters Π

```
1  M = DirectedMultigraph()
2  G ← {G_i ∈ Σ}                          // Generators container
3  C ← {C_j ∈ Σ}                          // Connectors container
4  while M.NODES = ∅ do
5      for σ ∈ G do
6          θ ← O.SampleParameters(σ)
7          τ ← σ(θ) // Estimatd, valid trajectories
8          if τ ≠ ∅ then
9              M.AddNode((σ, θ, τ), Cost(τ))
10         end
11     end
12 end
13 τ_start ← {x_start}
14 M.AddNode((∅, ∅, τ_start), 0) ; // Add start state as a node,
     with zero cost
15 while ¬M.HasPath(x_start, {x | ξ_goal(x) = 1}) do
       // Oracle selects next skill to apply
16     σ ← O.ChooseSkill(Σ, M)
17     θ ← O.SampleParameters(σ)
18     if σ ∈ C then
           // Apply connector skill
19         ξ_0, ξ_1 ← O.ChooseCondsToConnect(M)
20         τ ← σ(ξ_0, ξ_1, θ)
21         if τ ≠ ∅ then
22             M.AddEdge((σ, ξ_0, ξ_1, θ), Cost(τ))
23         end
24     else
           // Apply generator skill
25         τ ← σ(θ)
26         if τ ≠ ∅ then
27             M.AddNode((σ, θ, τ), Cost(τ))
28         end
29     end
30 end
   // Least-cost path to any goal state
31 return M.ShortestPath(x_start, {x | ξ_goal(x) = 1})
```

Main Algorithm Cont..

MOSAIC's main loop repeats the following until a path from start to goal exists:

1. Oracle decides: Generator or Connector? (based on graph structure) (line 16)
2. Oracle picks: which specific skill? (based on past performance + exploration) (line 16)
3. Oracle samples from predefined distribution: random parameters for that skill (e.g., where to push, which grasp) (line 17)
4. Skill executes:
   a. Generator (lines 24-29) → produces a trajectory freely
   b. Connector (lines 18-23) → oracle picks two nodes to link (line 19), then the connector attempts a trajectory between their full world states — robot + object configurations (line 20)
5. Simulator validates: Sapien rolls out the trajectory
   a. Valid → enters graph as node (line 27) or edge (line 22)
   b. Invalid → discarded
6. Check (line 15): path from start to goal exists?
   a. Yes → return least-cost path (line 31)
   b. No → repeat.

Note: For **stochastic skills**, MOSAIC generates multiple trajectories in parallel. If most succeed, the skill is considered reliable (low cost). If most fail, it's considered unreliable (high cost) → more on this later

Methodology

The Oracle (Skill Selection)

The oracle starts with no knowledge of which skills are good. It learns from trial and error during search.

Note: the oracle is modular and swappable (e.g., could be an LLM). The paper proposes a domain-independent statistical oracle.

Two-stage selection:

Stage 1 — Choose skill type:

- Oracle checks the node-to-edge ratio of the graph
- Many islands, few links → prioritize Connectors
- Few islands → prioritize Generators

Stage 2 — Choose specific skill within that type:

- Each candidate scored: $U(\sigma_i) = \alpha \cdot s_{\sigma\_i} + (1-\alpha) \cdot \sqrt{(\ln(\Sigma(t_{\sigma\_j}+1))/(t_{\sigma\_i}+1))} + n$
- In plain terms: skills that have worked well score high (exploitation), skills we haven't tried much get a bonus (exploration), noise prevents repetitive cycling
- Highest score wins

Methodology

The Oracle (Skill Selection Cont...)

$$U(\sigma_i) = \alpha \cdot s_{\sigma\_i} + (1-\alpha) \cdot \sqrt{(\ln(\Sigma(t_{\sigma\_j}+1))/(t_{\sigma\_i}+1))} + n$$

- Where:
  - $\alpha \cdot s_{\sigma\_i}$ is the exploitation term with s_{σ_i} is the success rate of skill σ_i , tracked during the current search
    - Higher success rate → higher score → more likely to be selected.
  - $(1-\alpha) \cdot \sqrt{(\ln(\Sigma(t_{\sigma\_j}+1))/(t_{\sigma\_i}+1))}$ is the exploration term with t_{σ_i} is how many times skill σ_i has been invoked
    - If a skill has been used rarely relative to others, the denominator is small, making this term large → under-explored skills get a bonus
  - $n \sim N(0,1)$ is the Gaussian noise for stochasticity, prevents deterministic cycling over the same skills
  - α is the scalar that controls the balance between exploitation and exploration.

Methodology

The Oracle (Node Pair Selection)

When a connector is selected, the oracle must also decide: which two nodes to attempt linking?

Most often, it selects two existing nodes using one of four modes via random number draw:

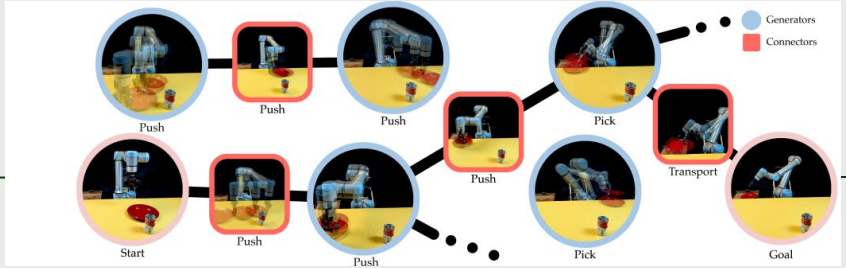
- Random: pick a node, connect to nearest neighbor
- Start bias: pick a node reachable from start, connect to nearest neighbor
- Goal bias: pick a node already connected to a goal-satisfying node, connect to nearest neighbor
- Unification: bridge nodes reachable from start with nodes reaching goal

Otherwise, the connector attempts to connect a node directly to the goal condition

- (e.g., Transport attempts to place the plate in the bin).

Nearest neighbor = node whose robot + object configuration is most similar (measured by pose difference).

Failed attempts are penalized by inflating pair distance, discouraging re-selection!



Putting it all together, the oracle first used generators to create islands of competence (circles) — multiple Push nodes and a Pick node.

It then selected connectors (squares) to link them, eventually unifying a path from Start to Goal via Push → Push → Push → Pick → Transport

Theoretical Guarantees

Probabilistic Completeness (PC):

- Definition: Solution is feasible if it can be decomposed into trajectory segments where every segment can be generated by some skill in the library
- Definition: An algorithm is probabilistically complete if: as iterations $k \to \infty$, the probability of finding a feasible solution (if one exists) approaches 1

MOSAIC is PC because two conditions are met as $k \to \infty$:

1. All generators are invoked with every parameter configuration → all possible islands are discovered
2. All connectors attempt to link every disconnected node with all parameters → all possible connections are tried

This is ensured because:

- The oracle assigns non-zero probability to selecting any skill
- Parameters are sampled randomly
- Therefore all skill-parameter combinations are eventually explored

Note: PC is relative to the skill library — MOSAIC guarantees finding a solution only if one is achievable with the given skills!

MOSAIC returns the least-cost path found in the current graph, but this is not guaranteed to be globally optimal!

Experiment Setup

Skill Library:

MOSAIC is tested with four skills. All rolled out in the Sapien physics engine to verify feasibility during planning:

- Push (G+C): uses a learned diffusion policy to generate pushing motions up to 25cm (~70% success rate)
  - Trained on 10,000 examples
  - Both versions use a motion planner to reach pre-push configurations
  - As Generator: randomly pushes objects to new positions, creating new world states
  - As Connector: links full world states across nodes
    i. Diffusion policy moves the object between specific poses
    ii. Motion planner additionally matches the robot configuration at the destination node
- Executes step-by-step using simulator as feedback
- Pick (G): computes antipodal grasps by scoring surface normal alignment with gripper direction, then verifies IK feasibility
  - Executes screw-based motion through pre-grasp → grasp → retract
- Transport (C): moves grasped objects to goal using motion planning (RRT-Connect)
  - Fails if object not grasped or goal unmet
- Rearrange (C, Scenario 3 only): combines pick-and-place and push to reposition multiple objects

Experiment Setup

Scenarios:

Three scenarios of increasing complexity, all sharing one goal: "plate in bin":

- Scenario 1: Basic Transport — plate deep on table, no obstacles
  - Must push to edge, pick from side, place in bin
- Scenario 2: Transport in Clutter — static obstacles added around the plate
  - Must navigate clutter without collisions
- Scenario 3: Transport Among Movable Objects — second movable object (chips can) may obstruct plate
  - Robot can work around it or manipulate it

(a) Transport.

(b) Transport in Clutter.

(c) Transport Among Movable Objects.

Experiment Setup

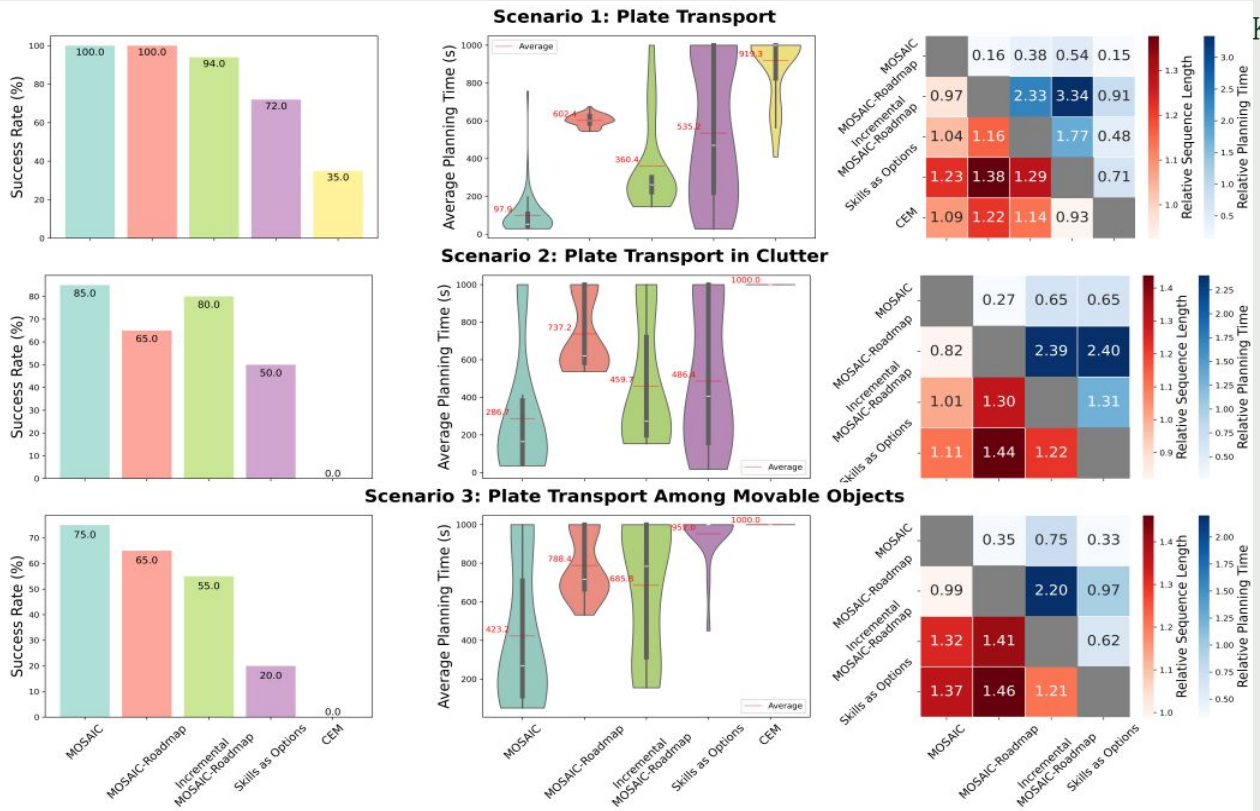
Four baselines:

- Skills as Options: sequential skill chaining via BFS
  - each skill starts where the previous ended (goal-directed)
- CEM: receding horizon planning — samples skill sequences, refines, executes first skill, replans
- MOSAIC-Roadmap: PRM-inspired — freely generates trajectories, connects via roadmap, uses Dijkstra's
  - Reports failure if no path found
- Incremental MOSAIC-Roadmap: extends above by iteratively adding nodes/edges
  - Closest to MOSAIC but lacks adaptive oracle guidance

Evaluation Metrics:

- Success rate: fraction of trials reaching a goal-satisfying state
- Plan length: number of skills in solution (proxy for solution quality)
- Planning time: algorithm efficiency

Experiment Analysis



Key Takeaways:

- MOSAIC achieves the highest success rates across all scenarios, maintaining performance as complexity increases
- Goal-directed methods (Skills as Options, CEM) degrade significantly with complexity
- The oracle is the key differentiator — comparing MOSAIC to Incremental MOSAIC-Roadmap (closest baseline, lacks adaptive oracle) shows the value of adaptive guidance
- MOSAIC achieves competitive or faster planning times than all baselines
- Hardware validation on UR10e trended similarly

Limitations

Paper's Acknowledgement:

- Requires access to a high-fidelity physics simulator — sim-to-real gap is a concern
- Skill library is provided, not learned — MOSAIC can only solve tasks achievable with the given skills
- Assumes full observability — partial observability listed as future work
- Domain-independent oracle is general but potentially suboptimal — task-aware oracles could be more efficient

Inferable:

- Every skill invocation requires a simulation rollout — planning times in the hundreds of seconds
- The experiments only test up to two movable objects — performance with significantly more objects or a larger skill library remains untested
- No optimality guarantees — only probabilistic completeness is proven

Conclusion

Three key contributions:

- Skills actively guide planning toward regions where they succeed, rather than relying on goal-directed chaining
- Modular architecture (generators, connectors, physics simulation) avoids hand-coded skill preconditions and effects
- MOSAIC consistently achieves high success rates with competitive planning times across increasingly complex tasks

Future directions:

- More sophisticated oracle modules
- Integrating foundation models for high-level reasoning
- Extending to partial observability
- Better utilizing parallelization of physics simulators

Thank You!