Text2Motion: From Natural Language Instructions to Feasible Plans

Project page: sites.google.com/stanford.edu/text2motion

Kevin Lin*†, Christopher Agia*†‡, Toki Migimatsu†, Marco Pavone‡ and Jeannette Bohg†

† Department of Computer Science, Stanford University, California, U.S.A.

‡ Department of Aeronautics & Astronautics, Stanford University, California, U.S.A.

Email: {kevinlin0,cagia,takatoki,pavone,bohg}@stanford.edu

***Abstract*—We propose Text2Motion, a language-based planning framework enabling robots to solve sequential manipulation tasks that require long-horizon reasoning. Given a natural language instruction, our framework constructs both a task- and motion-level plan that is verified to reach inferred symbolic goals. Text2Motion uses feasibility heuristics encoded in Q-functions of a library of skills to guide task planning with Large Language Models. Whereas previous language-based planners only consider the feasibility of individual skills, Text2Motion actively resolves geometric dependencies spanning skill sequences by performing geometric feasibility planning during its search. We evaluate our method on a suite of problems that require long-horizon reasoning, interpretation of abstract goals, and handling of partial affordance perception. Our experiments show that Text2Motion can solve these challenging problems with a success rate of 82%, while prior state-of-the-art language-based planning methods only achieve 13%. Text2Motion thus provides promising generalization characteristics to semantically diverse sequential manipulation tasks with geometric dependencies between skills.**

I. Introduction

Long-horizon robot planning is traditionally formulated as a symbolic and geometric reasoning problem, where the symbolic reasoner is supported by a formal logic representation (e.g. first-order logic [1]). Such systems can generalize within the logical planning domain specified by experts. However, many desirable properties of plans that can be specified in language may be cumbersome or impossible to represent in formal logic. Examples include user intent or preferences, and reasoning beyond what is known about the environment.

The emergence of *Large Language Models* (LLMs) [4] as a task-agnostic reasoning module presents a promising pathway to general robot planning capabilities. Several recent works [3, 9, 13] capitalize on their ability to perform robot planning without needing to manually specify symbolic planning domains. Nevertheless, these prior approaches are challenged when facing tasks that require long-horizon symbolic and geometric reasoning. In this paper, we ask: how can we verify the correctness and feasibility of LLM-generated plans?

We propose **Text2Motion**, a language-based planning framework that interfaces an LLM with a library of learned skills and a geometric feasibility planner [2] to solve complex sequential manipulation tasks (Fig. 1). Our contributions are two-fold: (i) a hybrid LLM planner that synergistically integrates shooting- and search-based planning strategies to construct geometrically feasible plans for tasks not seen by the skills during training; and (ii) a plan termination method that infers goal states from a natural language instruction to verify the completion of plans. We find that our hybrid planner achieves a success rate of 82% on a suite of challenging table top manipulation tasks, while prior language-based planning methods achieve a 13% success rate.

Toyota Research Institute provided funds to support this work. This work was also supported by the National Aeronautics and Space Administration (NASA) under the Innovative Advanced Concepts (NIAC) program.

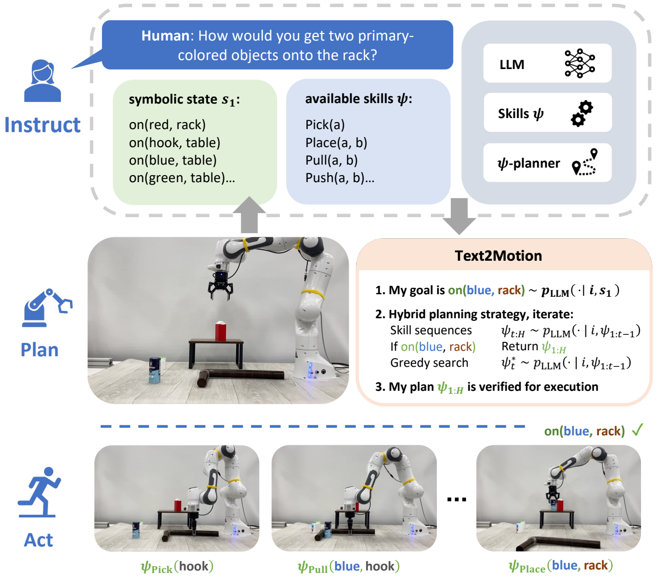
Instruct
Human: How would you get two primary-colored objects onto the rack?
symbolic state s_1:
on(red, rack)
on(hook, table)
on(blue, table)
on(green, table)...
available skills ψ:
Pick(a)
Place(a, b)
Pull(a, b)
Push(a, b)...
LLM
Skills ψ
ψ-planner
Plan
Text2Motion
1. My goal is on(blue, rack) $\sim p_{\mathrm{LLM}}(\cdot \mid i, s_1)$
2. Hybrid planning strategy, iterate:
Skill sequences $\psi_{t:H} \sim p_{\mathrm{LLM}}(\cdot \mid i, \psi_{1:t-1})$
If on(blue, rack) Return $\psi_{1:H}$
Greedy search $\psi_t^* \sim p_{\mathrm{LLM}}(\cdot \mid i, \psi_{1:t-1})$
3. My plan $\psi_{1:H}$ is verified for execution
on(blue, rack) ✓
Act
ψ_{Pick}(hook)
ψ_{Pull}(blue, hook)
...
ψ_{Place}(blue, rack)

Fig. 1. To carry out the instruction "get two primary-colored objects onto the rack", the robot must apply symbolic reasoning over the scene description and natural language instruction to deduce what skills should be executed to acquire a second primary-colored object, after noticing that a red object is already on the rack (i.e. on(red, rack)). It must also apply geometric reasoning to ensure that skills are sequenced in a manner that is likely to succeed. Unlike prior work [3, 9] that myopically executes skills a the current timestep, Text2Motion constructs plans of skills and coordinates their dependencies with geometric feasibility planning [2]. Upon planning the skill sequence Pick(hook), Pull(blue, hook), Pick(blue), Place(blue, rack), our method computes a grasp position on the hook that enables pulling the blue object in a way that is conducive to later picking up the blue object.

II. Problem Setup

We aim to solve long-horizon sequential manipulation problems that require symbolic and geometric reasoning from an instruction i expressed in natural language.

A. *LLM and skill library*

We assume access to an LLM and a library of skills $\mathcal{L}^{\psi} = \{\psi^1, \ldots, \psi^N\}$. Each skill ψ consists of a policy $\pi(a|s)$, parameterized manipulation primitive $\phi(a)$ [6], language description, Q-function $Q^{\pi}(s, a)$, and dynamics model

$T^\pi(s'|s,a)$, and is associated with a contextual bandit, or a single-timestep Markov Decision Process (MDP):

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \rho), \tag{1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(s'|s,a)$ is the transition model, $R(s,a,s')$ is the binary reward function, and $\rho(s)$ is the initial state distribution. When a skill ψ is executed, an action $a \in \mathcal{A}$ is sampled from its policy $\pi(a|s)$ and fed to its primitive $\phi(a)$ which consumes the action and executes a series of motor commands on the robot. If the skill succeeds, it receives a binary reward of r (or $\neg r$ if it fails). We refer to policy actions $a \in \mathcal{A}$ as primitive *parameters*, which, depending on the skill, can represent grasp poses, placement locations, pulling or pushing distances (Appx. A-A).

B. The planning objective

Our objective is to find a plan in the form of a sequence of skills $\psi_{1:H}$ that is both likely to satisfy the instruction i and can be successfully executed in the environment. This objective can be expressed as the joint probability of skill sequence $\psi_{1:H}$ and binary rewards $r_{1:H}$ given the instruction i and initial state s_1:

$$\begin{aligned} &p(\psi_{1:H}, r_{1:H} \mid i, s_1) \\ &\quad = p(\psi_{1:H} \mid i, s_1)\, p(r_{1:H} \mid i, s_1, \psi_{1:H}). \end{aligned} \tag{2}$$

Here, $p(\psi_{1:H} \mid i, s_1)$ considers the probability that the skill sequence $\psi_{1:H}$ will satisfy the instruction i from a symbolic perspective. However, a symbolically correct skill sequence may fail during execution due to kinematic constraints of the robot or geometric dependencies spanning the skill sequence. We must also consider the *success probability* of the skill sequence $\psi_{1:H}$ captured by $p(r_{1:H} \mid i, s_1, \psi_{1:H})$, which depends on the parameters $a_{1:H}$ fed to the underlying sequence of primitives $\phi_{1:H}$ that control the robot's motion:

$$p(r_{1:H} \mid i, s_1, \psi_{1:H}) = p(r_{1:H} \mid s_1, a_{1:H}). \tag{3}$$

This represents the probability that skills $\psi_{1:H}$ achieve rewards $r_{1:H}$ when executed from initial state s_1 with parameters $a_{1:H}$, which is independent of the instruction i. If just one skill fails (reward $\neg r$), then the entire plan fails.

C. Geometric feasibility planning

The role of geometric feasibility planning is to maximize the success probability (Eq. 3) of a skill sequence $\psi_{1:H}$ by finding an optimal set of parameters $a_{1:H}$ for the underlying primitive sequence $\phi_{1:H}$. This process is essential for finding plans that maximize the overall planning objective in Eq. 2.

In our experiments, we leverage Sequencing Task-Agnostic Policies (STAP) [2]. STAP resolves geometric dependencies across the skill sequence $\psi_{1:H}$ by maximizing the product of step reward probabilities of parameters $a_{1:H}$:

$$a^*_{1:H} = \arg\max_{a_{1:H}} \ \mathrm{E}_{s_{2:H}} \left[\prod_{t=1}^{H} p(r_t \mid s_t, a_t) \right], \tag{4}$$

where future states $s_{2:H}$ are predicted by dynamics models $s_{t+1} \sim T^{\pi_t}(\cdot|s_t, a_t)$. The reward probability $p(r_t \mid s_t, a_t)$ is equivalent to the Q-function $Q^{\pi_t}(s_t, a_t)$ for ψ_t in a contextual bandit setting with binary rewards (Eq. 1). The success probability of the optimized skill sequence $\psi_{1:H}$ is approximated by the product of Q-functions evaluated from initial state s_1 along a sampled trajectory $s_{2:H}$ with parameters $a^*_{1:H}$:

$$p(r_{1:H} \mid s_1, a_{1:H}) \approx \prod_{t=1}^{H} Q^{\pi_t}(s_t, a^*_t). \tag{5}$$

III. Methods

The core idea of this paper is to ensure the geometric feasibility of an LLM task plan—and thereby its correctness—by predicting the success probability (Eq. 3) of learned skills that are sequenced according to the task plan. In the following sections, we outline two strategies for planning with LLMs and learned skills: a shooting-based planner and a search-based planner. We then introduce the full planning algorithm, **Text2Motion**, which synergistically integrates the strength of both strategies. These strategies represent different ways of maximizing the overall planning objective in Eq. 2.

A. Goal prediction

Plans with high overall objective scores (Eq. 2) are not guaranteed to satisfy their instruction. Thus, the first step in all planning strategies is to convert the language instruction into a goal condition that can be checked against a candidate sequence of skills. Given an instruction i, a set of objects O in the scene, and a library of predicate classifiers $\mathcal{L}^\chi = \{\chi^1, \dots, \chi^M\}$, we use the LLM to predict a set of symbolic goal states $\mathcal{G} = \{g_1, \dots, g_j\}$ that would satisfy the instruction. Each predicate classifier χ is a binary-valued function over object poses that evaluates whether a spatial relationship exists in the scene (details in Appx. B-A). We define a satisfaction function $F^{\mathcal{G}}_{\text{sat}}(s) : \mathcal{S} \to \{0, 1\}$ that converts the geometric state s into a symbolic state using predicate classifiers $\mathcal{L}^\chi$ and checks if any goal $g \in \mathcal{G}$ predicted by the LLM is satisfied. A sequence of skills $\psi_{1:H}$ is said to satisfy the instruction i *iff*:

$$\exists\, s \in s_{2:H+1} : F^{\mathcal{G}}_{\text{sat}}(s) = 1, \tag{6}$$

where the future states $s_{2:H+1}$ are predicted by the geometric feasibility planner (see Sec. II-C). If $F^{\mathcal{G}}_{\text{sat}}(s_t)$ evaluates to 1 for a geometric state s_t at timestep $t \le H+1$, then the planner returns the subsequence of skills $\psi_{1:t-1}$ for execution.

B. Shooting-based planning

We propose a first planner, termed **SHOOTING** (see Fig. 2) that takes a single-step approach to maximizing the overall planning objective in Eq. 2. **SHOOTING** queries the LLM once to generate K candidate skill sequences $\{\psi^1_{1:H}, \dots, \psi^K_{1:H}\}$ in an open-ended fashion. Each candidate skill sequence is processed by the geometric feasibility planner which returns an estimate of the sequence's success probability (Eq. 5) and its predicted future state trajectory $s_{2:H}$. Skill sequences that satisfy the goal condition (Eq. 6) are added to a candidate set. If the candidate set is not empty, **SHOOTING** returns the skill sequence with the highest success probability, or raises a `planning failure` otherwise.

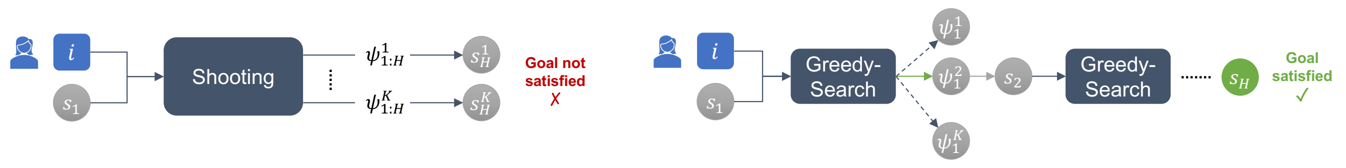

Fig. 2. **SHOOTING and GREEDY-SEARCH planning overview**. Both SHOOTING and GREEDY-SEARCH planning algorithms use the LLM to predict the set of valid goal state propositions given the user's natural language instruction and the current state. This predicted goal proposition is used to decide when the goal is satisfied and planning can be terminated. **Left:** The SHOOTING method uses the LLM to propose entire plans first and then runs geometric feasibility planning afterwards. As shown in the experiments, this approach fails when the space of candidate task plans is large but few skill sequences are geometrically feasible. **Right:** In the GREEDY-SEARCH planning algorithm, the LLM is first used to propose K candidate skills with the top LLM scores. The geometric feasibility planner then evaluates the feasibility of each candidate skill, and the one with the highest product of LLM and geometric feasibility scores is selected. The successor state of this skill is predicted by the geometric feasibility planner's dynamics model. If the predicted state does not satisfy the goal propositions, then it is given to the LLM to plan the next skill. GREEDY-SEARCH interleaves LLM planning with geometric feasibility planning at each iteration. If the goal propositions are satisfied, then the planner returns the skill sequence.

C. Search-based planning

We propose a second planner, **GREEDY-SEARCH**, which, at each planning iteration, ranks candidate skills predicted by the LLM and adds the top scoring skill to the running plan. This iterative approach can be described as a decomposition of the planning objective in Eq. 2 by timestep t:

$$p(\psi_{1:H}, r_{1:H} \mid i, s_1) = \prod_{t=1}^{H} p(\psi_t, r_t \mid i, s_1, \psi_{1:t-1}, r_{1:t-1}). \tag{7}$$

We define the joint probability of ψ_t and r_t in Eq. 7 to be the skill score S_{skill}:

$$S_{\text{skill}}(\psi_t) = p(\psi_t, r_t \mid i, s_1, \psi_{1:t-1}, r_{1:t-1}),$$

which we factor using conditional probabilities:

$$S_{\text{skill}}(\psi_t) = p(\psi_t \mid i, s_1, \psi_{1:t-1}, r_{1:t-1}) \, p(r_t \mid i, s_1, \psi_{1:t}, r_{1:t-1}). \tag{8}$$

Each planning iteration t of **GREEDY-SEARCH** is responsible for finding the skill ψ_t that maximizes the skill score (Eq. 8).

Skill usefulness. The first factor of Eq. 8 captures the *usefulness* of a skill generated by the LLM for satisfying the instruction. We define the skill usefulness score S_{llm}:

$$S_{\text{llm}}(\psi_t) = p(\psi_t \mid i, s_1, \psi_{1:t-1}, r_{1:t-1}) \tag{9}$$
$$\approx p(\psi_t \mid i, s_{1:t}, \psi_{1:t-1}). \tag{10}$$

In Eq. 10, the probability of the next skill ψ_t (Eq. 9) is cast in terms of the predicted state trajectory $s_{2:t}$ of the running plan $\psi_{1:t-1}$, and is thus is independent of prior rewards $r_{1:t-1}$.

At each planning iteration t, we optimize $S_{\text{llm}}(\psi_t)$ by querying an LLM to generate K candidate skills $\{\psi^1_t, \dots, \psi^K_t\}$. We then compute the usefulness scores $S_{\text{llm}}(\psi^k_t)$ by summing the token log-probabilities of each skill's language description. These scores represent the likelihood that ψ^k_t is the correct skill to execute from the LLMs perspective to satisfy instruction i.

Skill feasibility. The second factor of Eq. 8 captures the feasibility of a skill generated by the LLM. We define the skill feasibility score S_{geo}:

$$S_{\text{geo}}(\psi_t) = p(r_t \mid i, s_1, \psi_{1:t}, r_{1:t-1}) \tag{11}$$
$$\approx Q^{\pi_t}(s_t, a^*_t), \tag{12}$$

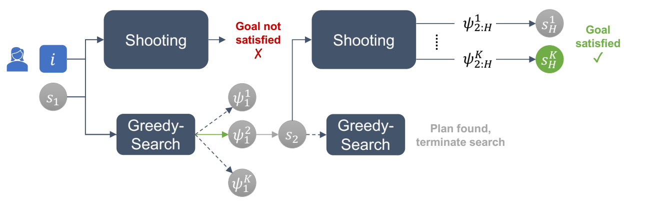

Fig. 3. **Proposed hybrid planner**. After predicting goals for a given instruction, Text2Motion iterates the process: i) invoke SHOOTING to plan full skill sequences, and if no goal-reaching plan is found, ii) take a GREEDY-SEARCH step and check if executing the selected "best" skill would reach the goal. Note that the entire planning process occurs before execution. See Fig. 2 for details on the GREEDY-SEARCH and SHOOTING planners.

where we approximate Eq. 11 by the Q-value evaluated at the predicted future state s_t with the optimized parameter a^*_t, both of which are computed by the geometric feasibility planner.

The skill feasibility score (Eq. 12) and skill usefulness score (Eq. 10) are then multiplied to produce the overall skill score (Eq. 8) for each of the K candidate skills $\{\psi^1_t, \dots, \psi^K_t\}$. Invalid skills as determined by Sec. III-E are filtered-out of the candidate set. Of the remaining skills, the one with the highest skill score ψ^*_t is added to the running plan $\psi_{1:t-1}$. If the geometric state s_{t+1} that results from skill ψ^*_t satisfies the goal condition (Eq. 6), the skill sequence $\psi_{1:t}$ is returned for execution. Otherwise, s_{t+1} is used to initialize planning iteration $t+1$. The process repeats until the planner returns or a maximum search depth $d_{\max}$ is met (i.e. `planning failure`).

D. Hybrid planning with Text2Motion

We present **Text2Motion**, a hybrid planning algorithm that inherits the strengths of both shooting-based and search-based planning strategies. See visualization in Fig. 3.

E. Out-of-distribution detection

During planning, the LLM may propose skills that are out-of-distribution (OOD) given the current state s_t and optimized parameter a^*_t. We consider a skill ψ_t to be OOD if the variance of its associated Q-value (Eq. 12) predicted by an ensemble [12] exceeds a calibrated threshold ϵ^{ψ_t}:

$$F_{\text{OOD}}(\psi_t) = \mathbb{1}\left(\text{Var}_{i\sim 1:B}\left[Q^{\pi_t}_i(s_t, a^*_t)\right] \geq \epsilon^{\psi_t}\right),$$

where $\mathbb{1}$ is the indicator function and B is the ensemble size. We refer to Appx. A-B for details on calibrating OOD thresholds ϵ^{ψ} for ensembles of Q-functions.

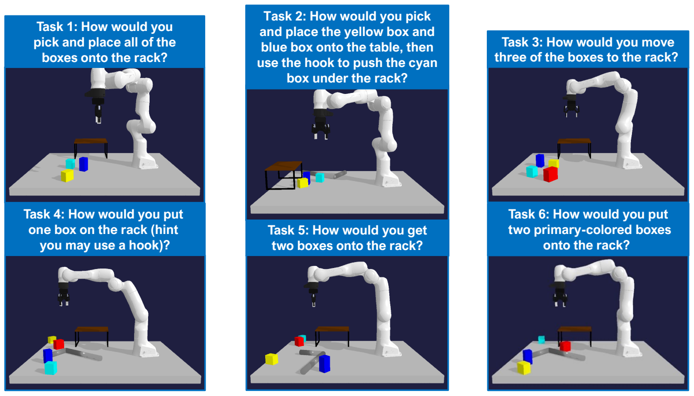
Task 1: How would you pick and place all of the boxes onto the rack?
Task 2: How would you pick and place the yellow box and blue box onto the table, then use the hook to push the cyan box under the rack?
Task 3: How would you move three of the boxes to the rack?
Task 4: How would you put one box on the rack (hint you may use a hook)?
Task 5: How would you get two boxes onto the rack?
Task 6: How would you put two primary-colored boxes onto the rack?

Fig. 4. **TableEnv Manipulation evaluation task suite**. We evaluate the performance of all methods on a task suite based on the above manipulation domain. The tasks considered vary in terms of difficulty and each contains a subset of three properties: i) (**LH**: Tasks 1, 2, 3, 5, 6) long-horizon tasks require skill sequences $\psi_{1:H}$ of length six or greater to solve; ii) (**LG**: Tasks 3, 4, 5, 6) lifted goals are expressed over object classes rather than object instances; iii) (**PAP**: Tasks 4, 5, 6) partial affordance perception, where skill affordances cannot be perceived by the LLM solely from the spatial relationships described in the initial state s_1. During evaluation, we randomize the geometric parameters of each task across 10 random seeds.

IV. Experiments

We conduct experiments to test the following hypothesis: (**H1**) geometric feasibility planning is necessary for solving TAMP-like problems specified by natural language with LLMs and robot skills; (**H2**) **Text2Motion**'s hybrid planner inherits the strengths of search- and shooting-based strategies.

A. Baselines

We compare **Text2Motion** with a series of language-based planners, including **SHOOTING** and **GREEDY-SEARCH**.

SAYCAN-GS: We implement a cost-considerate variant of SayCan [3] with a module dubbed **GENERATOR-SCORER** (GS). At each timestep t, SayCan ranks *all possible* skills by $p(\psi_t \mid i, \psi_{1:t-1}) \cdot V^{\psi_t}(s_t)$, before executing the top scoring skill (Scorer). However, the cost of ranking skills scales unfavorably with the number of scene objects $\mathcal{O}$ and skills in library $\mathcal{L}^{\psi}$. **SAYCAN-GS** limits the pool of skills considered in the ranking process by querying the LLM for the K most useful skills $\{\psi_t^1, \ldots, \psi_t^K\} \sim p(\psi_t \mid i, \psi_{1:t-1})$ (Generator) before engaging Scorer. Execution terminates when the score of the `stop` "skill" is larger than the other skills.

INNERMONO-GS: We implement the *Object + Scene* variant of Inner Monologue [9] by providing task-progress scene context in the form of the environment's symbolic state. We acquire **INNERMONO-GS** by equipping [9] with **GENERATOR-SCORER** for cost efficiency. LLM skill likelihoods are equivalent to those from **SAYCAN-GS** except they are now also conditioned on the visited state history $p(\psi_t \mid i, s_{1:t}, \psi_{1:t-1})$.

B. Evaluation and metrics

We construct a suite of evaluation tasks in a table-top manipulation domain (Fig. 4). **Text2Motion**, **SHOOTING**, and **GREEDY-SEARCH** produce a successful plan if, upon execution, the plan reaches a final state s_H that satisfies the instruction i. We only execute a plan if the final dynamics predicting state s_H satisfies $F_{\text{sat}}^{\mathcal{G}}$ (Sec. III-A). Both methods perform closed-loop execution of the skill sequence $\psi_{1:H}$, calling [2] to perform skill sequence optimization on the skills $\psi_{t+1:H}$ after executing skill ψ_t. We do not perform task-level replanning. **SAYCAN-GS** and **INNERMONO-GS** are myopic agents that execute the next best admissible skill ψ_t at each timestep. Hence, we evaluate them in a closed-loop manner for a maximum of $d_{\max}$ steps. We mark a run as a success if the method issues the `stop` skill and the current state satisfies the ground-truth goal.

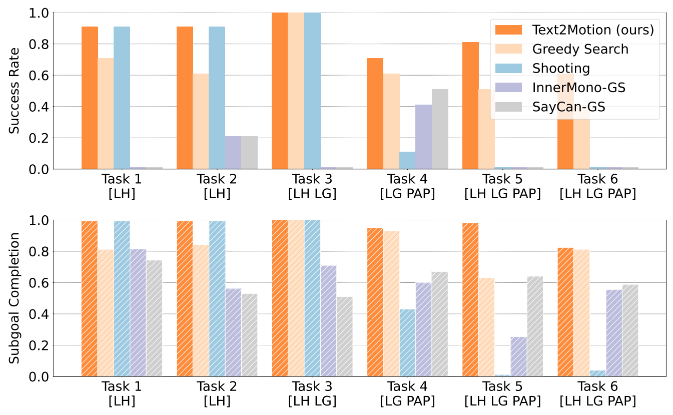

Fig. 5. **Results on the TableEnv manipulation domain** with 10 random seeds for each task. **Top:** Our method (Text2Motion) significantly outperforms all baselines on tasks involving partial affordance perception (Task 4, 5, 6). For tasks without partial affordance perception, the methods that use geometric feasibility planning (Text2Motion, SHOOTING, GREEDY-SEARCH) convincingly outperform the methods (SAYCAN-GS and INNERMONO-GS) that do not. We note that SHOOTING performs well on the tasks without partial affordance perception as it has the advantage of outputting *multiple* goal-reaching candidate plans and selecting the one with the highest execution success probability. **Bottom:** Methods without geometric feasibility planning tend to have high sub-goal completion rates but very low success rates. This divergence arises because it is possible to make progress on tasks without resolving geometric dependencies in the earlier timesteps; however, failure to account for geometric dependencies results in failure of the overall task.

V. Results

The results are in Fig. 5. We find that geometric feasibility planning is indeed required to solve TAMP-like problems with LLMs (**H1**) and that hybrid planning integrates the strengths of search-based and shooting-based methods (**H2**).

VI. Conclusion

We propose Text2Motion, a language-based planner that combines LLMs, learned skills, and skill sequence optimization to solve robot manipulation tasks with geometric dependencies. Text2Motion constructs a plan at the task and skill levels, and verifies its satisfaction of a natural language instruction by testing the planned states against inferred goal propositions *prior to execution in the real world*. It uses value function heuristics to guide LLM task planning during search and optimizes the running sequence of skills to resolve long-horizon action dependencies. Results show the importance of geometric feasibility planning, search-based planning and plan termination via inferred symbolic constraints for solving TAMP-like tasks from a natural language instruction.

References

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[2] Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. Stap: Sequencing task-agnostic policies. *arXiv preprint arXiv:2210.12250*, 2022.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Javier Felip, Janne Laaksonen, Antonio Morales, and Ville Kyrki. Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks. *Robotics and Autonomous Systems*, 61(3):283–296, 2013.

[7] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

[9] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

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

[11] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[12] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[13] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[14] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[15] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[16] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

[17] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

Overview

The appendix offers additional details with respect to the implementation of Text2Motion and language planning baselines (Appx. A), the experiments conducted (Appx. B), and the **real-world demonstrations** (Appx. C). Qualitative results are made available at sites.google.com/stanford.edu/text2motion.

Appendix A
Implementation Details

Text2Motion performs an iterative search to construct skill sequences that are feasible for the robot to execute in the environment. The iterative search procedure relies on four core components: 1) a library of learned robot skills, 2) a method for detecting when a skill is out-of-distribution (OOD), 3) a large language model (LLM) to perform task-level planning, and 4) a geometric feasibility planner that is compatible with the learned robot skills. The **SHOOTING** baseline also uses each of the above components in an alternative planning strategy, while **SAYCAN-GS** and **INNERMONO-GS** are myopic agents that do not use geometric feasibility planning. We provide implementation details of these components in the following subsections.

A. Learning robot skills and dynamics

Skill library overview: All evaluated language planners interface an LLM with a library of robot skills $\mathcal{L} = \{\psi^1, \dots, \psi^N\}$. Each skill ψ has a language description (e.g. Pick(a)) and is associated with a parameterized manipulation primitive [6] $\phi(a)$. A primitive $\phi(a)$ is controllable via its *parameter* a which determines the motion [11] of the robot's end-effector through a series of waypoints. For each skill ψ, we train a policy $\pi(a|s)$ to output parameters $a \in \mathcal{A}$ that maximize primitive's $\phi(a)$ probability of success in a contextual bandit setting (Eq. 1) with a skill-specific binary reward function $R(s, a, s')$. We also train an ensemble of Q-functions $Q^{\pi}_{1:B}(s, a)$ and a dynamics model $T^{\pi}(s'|s, a)$ for each skill, both of which are required for geometric feasibility planning. We discuss the calibration of Q-function ensembles for OOD detection of skills in Appdx A-B.

We learn four manipulation skills to solve tasks in simulation and in the real-world: ψ^{Pick}, ψ^{Place}, ψ^{Pull}, ψ^{Push}. Only a single policy per skill is trained, and thus, the policy must learn to engage the primitive over objects with differing geometries (e.g. π^{Pick} is used for both Pick(box) and Pick(hook)). The state space $\mathcal{S}$ for each policy is defined as the concatenation of geometric state features (e.g. pose, size) of all objects in the scene, where the first n object states correspond to the n skill arguments and the rest are randomized. For example, the state for the skill Pick(hook) would have be a vector of all objects' geometric state features with the first component of the state corresponding to the hook.

Parameterized manipulation primitives: We describe the parameters a and reward function $R(s, a, s')$ of each parameterized manipulation primitive $\phi(a)$ below. A collision with a non-argument object constitutes an execution failure for all skills, and as a result, the policy receives a reward of 0. For example, π^{Pick} would receive a reward of 0 if the robot collided with box during the execution of Pick(hook).

- Pick(obj): $a \sim \pi_{\text{Pick}}(a|s)$ denotes the grasp pose of obj *w.r.t* the coordinate frame of obj. A reward of 1 is received if the robot successfully grasps obj.
- Place(obj, rec): $a \sim \pi_{\text{Place}}(a|s)$ denotes the placement pose of obj *w.r.t* the coordinate frame of rec. A reward of 1 is received if obj is stably placed on rec.
- Pull(obj, tool): $a \sim \pi_{\text{Pull}}(a|s)$ denotes the initial position, direction, and distance of a pull on obj with tool *w.r.t* the coordinate frame of obj. A reward of 1 is received if obj moves toward the robot by a minimum of $d_{\text{Pull}} = 0.05m$.
- Push(obj, tool, rec): $a \sim \pi_{\text{Push}}(a|s)$ denotes the initial position, direction, and distance of a push on obj with tool *w.r.t* the coordinate frame of obj. A reward of 1 is received if obj moves away from the robot by a minimum of $d_{\text{Push}} = 0.05m$ and if obj ends up underneath rec.

Dataset generation: All planning methods considered in this work rely on having accurate Q-functions $Q^{\pi}(s, a)$ to estimate the feasibility of skills proposed by the LLM. This places a higher fidelity requirement on the Q-functions than needed to learn a reliable policy, as the Q-functions must characterize both skill success (feasibility) and failure (infeasibility) at a given state. Because the primitives $\phi(a)$ reduce the horizon of policies $\pi(a|s)$ to a single timestep, and the reward functions are $R(s, a, s') = \{0, 1\}$, the Q-functions can be interpreted as binary classifiers of state-action pairs. Thus, we take a staged approach to learning the Q-functions Q^{π}, followed by the policies π, and lastly the dynamics models T^{π}.

Scenes in our simulated environment are instantiated from a symbolic specification of objects and spatial relationships, which together form a symbolic state $\mathfrak{s}$. The goal is to learn a *complete* Q-function that sufficiently covers the state-action space of each skill. We generate a dataset that meets this requirement in four steps: a) enumerate all valid symbolic states $\mathfrak{s}$; b) sample geometric scene instances s per symbolic state; c) uniformly sample actions over the action space $a \sim \mathcal{U}^{[0,1]^d}$; (d) simulate the states and actions to acquire next states s' and compute rewards $R(s, a, s')$. We slightly modify this sampling strategy to maintain a minimum success-failure ratio of 40%, as uniform sampling for more challenging skills like Pull and Push seldom emits a success (∼3%). We collect 1M (s, a, s', r) tuples per skill in a process that takes approximately twelve hours. Of the

1M samples, 800K of them are used for training ($\mathcal{D}_t$) while the remaining 200K are used for validation ($\mathcal{D}_v$). We use the same datasets to learn the Q-functions Q^π, policies π, and dynamics models T^π for each skill.

Model training: We train an ensemble of Q-functions with mini-batch gradient descent and logistic regression loss. Once the Q-functions have converged, we distill their returns into stochastic policies π through the maximum-entropy update [8]:

$$\pi^* \leftarrow \arg\max_{\pi} \mathrm{E}_{(s,a)\sim\mathcal{D}_t}[\min(Q^\pi_{1:B}(s,a)) \\ -\alpha \log \pi(a|s)].$$

Instead of evaluating the policies on $\mathcal{D}_v$, which contains states for which no feasible action exists, the policies are synchronously evaluated in an environment that exhibits only feasible states. This simplifies model selection and standardizes skill capabilities across primitives. We train a deterministic dynamics model per primitive using the forward prediction loss:

$$L_{\text{dynamics}}\left(T^\pi; \mathcal{D}_t\right) = \mathrm{E}_{(s,a,s')\sim\mathcal{D}_t} \, ||T^\pi(s,a) - s'||_2^2.$$

All Q-functions achieve precision and recall rates of over 95%. The average success rates of the converged skill policies over 100 evaluation episodes are: π_{Pick} with 99%, π_{Place} with 90%, π_{Pull} with 86%, π_{Push} with 97%. The dynamics models converge to within millimeter accuracy on the validation split.

Hyperparameters: The Q-functions, policies, and dynamics models are MLPs with hidden dimensions of size [256, 256] and ReLU activations. We train an ensemble of $B = 8$ Q-functions with a batch size of 128 and a learning rate of 1e-4 with a cosine annealing decay [14]. The Q-functions for pick, pull, and push converged on $\mathcal{D}_v$ in 3M iterations, while the Q-function for place required 5M iterations. We hypothesize that this is because classifying successful placements demands carefully attending to the poses and shapes of all objects in the scene so as to avoid collisions. The policies are trained for 250K iterations with a batch size of 128 and a learning rate of 1e-4, leaving all other parameters the same as [8]. The dynamics models are trained for 750K iterations with a batch size of 512 and a learning rate of 5e-4; only on successful transitions to avoid the noise associated with collisions and truncated episodes. The parallelized training of all models takes approximately 12 hours on an Nvidia Quadro P5000 GPU and 2 CPUs per job.

B. Out-of-distribution detection

The datasets described in Sec. A-A contain both successes and failures for symbolically valid skills like Pick(box). However, in interfacing robot skills with LLM task planning, it is often the case that the LLM will propose symbolically invalid actions, such as Pull(box, rack), that neither the skill policies, Q-functions, or dynamics models have observed in training. We found that a percentage of such out-of-distribution (OOD) queries would result in erroneously high Q-values, causing the skill to be selected. Attempting to execute such a skill leads to control exceptions or other undesirable events.

Whilst there are many existing techniques for OOD detection of deep neural networks, we opt to detect OOD queries on the learned Q-functions via deep ensembles due to their ease of calibration [12]. A state-action pair is classified as OOD if the empirical variance of the predicted Q-values is above a determined threshold:

$$F_{\text{OOD}}\left(\psi\right) = \mathbb{1}\left(\mathrm{Var}_{i\sim 1:B}\left[Q_i^\pi(s,a)\right] \geq \epsilon^\psi\right),$$

where each threshold ϵ^ψ is unique to skill ψ.

To determine the threshold value, we generate an a calibration dataset of 100K symbolically invalid states and actions for each skill. The process takes less than an hour on a single CPU as the actions are infeasible and need not be simulated in the environment (i.e. rewards are known to be 0). We compute the empirical variance the Q-ensembles across both the in-distribution and out-of-distribution datasets an bin the variances by predicted Q-value to produce a histogram. We observe that the histogram of variances produced from OOD queries was uniform across all predicted Q-values and was an order of magnitude large than the ensemble variances computed over in-distribution data. This simplified the selection of OOD thresholds, which we found to be: $\epsilon^{\text{Pick}} = 0.10$, $\epsilon^{\text{Place}} = 0.12$, $\epsilon^{\text{Pull}} = 0.10$, and $\epsilon^{\text{Push}} = 0.06$.

C. Skill sequence generation with LLMs

Text2Motion (which iteratively takes a greedy search step and shooting step), **GREEDY-SEARCH** and the myopic planning baselines **SAYCAN-GS** and **INNERMONO-GS** use `code-davinci-002` (Codex model [5]) to generate and score skills, while

SHOOTING queries `text-davinci-003` (variant of InstructGPT [15]) to directly output full skill sequences. We used a temperature setting of 0 for all our queries tasks.

To maintain consistency in the evaluation of various planners, we allow Text2Motion, **SAYCAN-GS**, and **INNERMONO-GS** to generate $K = 5$ skills ψ_h^k at each timestep h. Thus, every search iteration of **GREEDY-SEARCH** considers five possible extensions to the current running sequence of skills $\psi_{1:h-1}$. Similarly, **SHOOTING** generates $K = 5$ skill sequences.

As described in Sec. III-C, skills are selected at each timestep h via a combined usefulness and geometric feasibility score:

$$\begin{aligned} S_{\text{skill}}(\psi_h) &= S_{\text{llm}}(\psi_h) \cdot S_{\text{geo}}(\psi_h) \\ &= p(\psi_h \mid i, s_1, \psi_{1:h-1}) \cdot p(r_h \mid i, s_1, \psi_{1:h}), \end{aligned}$$

where Text2Motion, **GREEDY-SEARCH** and **SHOOTING** use geometric feasibility planning (details below in Appx. A-D) to compute $S_{\text{geo}}(\psi_h)$, while **SAYCAN-GS** and **INNERMONO-GS** use the current value function estimate $V^{\pi_h}(s_h) = \mathrm{E}_{a_h \sim \pi_h}[Q^{\pi_h}(s_h, a_h)]$. We find that in both cases, taking $S_{\text{llm}}(\psi_h)$ to be the SoftMax log-probability score produces a winner-takes-all effect, causing the planner to omit highly feasible skills simply because their associated log-probability was marginally lower than the LLM-likelihood of another skill. Thus, we dampen the SoftMax operation with a β-coefficient to balance the ranking of skills based on both feasibility and usefulness. We found $\beta = 0.3$ to work well in our setup.

D. Geometric feasibility planning

Given a skill sequence, we still need to select the underlying continuous parameters corresponding to each skill. For example, to execute a Pick(hook) skill, we would need to specify the exact underlying 3D location of the grasp. We note that Text2Motion is agnostic the method that fulfils the role of selecting continuous parameters for a sequence of skills. In our experiments we leverage Sequencing Task-Agnostic Policies (STAP) [2]. Specifically, we consider the PolicyCEM variant of STAP, where sampling-based optimization of the skill sequence's $\psi_{1:h}$ success probability is warm started with actions sampled from the policies $a_{1:h} \sim \pi_{1:h}$. We perform ten iterations of the Cross-Entropy Method (CEM) [16], sampling 10K trajectories at each iteration and selecting 10 elites to update the mean of the sampling distribution for the following iteration. The standard deviation of the sampling distribution is held constant at 0.3 for all iterations.

TABLE I

TableEnv manipulation task suite. We use the following shorthands as defined in the paper: LH: Long-Horizon, LG: Lifted Goals, PAP: Partial Affordance Perception.

Task ID	Properties	Instruction
Task 1	LH	How would you pick and place all of the boxes onto the rack?”
Task 2	LH + LG	How would you pick and place the yellow box and blue box onto the table, then use the hook to push the cyan box under the rack?”
Task 3	LH + PAP	How would you move three of the boxes to the rack?”
Task 4	LG + PAP	How would you put one box on the rack?”
Task 5	LH + LG + PAP	How would you get two boxes onto the rack?”
Task 6	LH + LG + PAP	How would you move two primary colored boxes to the rack?”

Appendix B
Experiment Details

We refer to Table. I for an overview of the tasks in the TableEnv Manipulation suite.

A. *Scene descriptions as symbolic states*

To provide scene context to Text2Motion and the baselines, we take a heuristic approach to converting a geometric state s into a basic symbolic state $\mathfrak{s}$. Symbolic states are comprised of three spatial relations: on(a, b), under(a, b), and inhand(a). inhand(a) = True when the object a’s z value is above a predefined height threshold. on(a, b) = True when i) a is above b (determined by checking if a’s axis-aligned bounding box is greater than b’x axis-aligned bounding box) ii) a’s bounding box intersects b’s bounding box iii) a is not inhand iv) the z distance between a and b is below a certain threshold. under(a, b) = True when a is not above b and a’s bounding box intersects b’s bounding box.

The proposed goal proposition prediction technique is also constrained to predict within this set of predicates. We highlight that objects are neither specified as within or beyond the robot workspace, as we leave it to the skill value functions to determine the feasibility of the primitives (Appx. A-A).

We note that planning in high-dimensional observation spaces is not the focus of this work. Thus, we assume knowledge of objects in the scene and use hand-crafted heuristics to detect spatial relations between objects. There exists several techniques to distill high-dimensional observations into scene descriptions, such as the one used in [17]. We leave exploration of these options to future work.

B. *In-Context Examples*

For all experiments and methods, we use the following in context examples.

```
Available scene objects: [’table’, ’hook’, ’rack’, ’yellow box’, ’blue box’, ’red box’]
Object relationships: [’inhand(hook)’, ’on(yellow box, table)’, ’on(rack, table)’, ’on(blue box, table)’]
Human instruction: How would you push two of the boxes to be under the rack?
Goal predicate set: [[’under(yellow box, rack)’, ’under(blue box, rack)’], [’under(blue box, rack)’, ’under(red
box, rack)’], [’under(yellow box, rack)’, ’under(red box, rack)’]]
Top 1 robot action sequences: [’push(yellow box, hook, rack)’, ’push(red box, hook, rack)’]
```

```
Available scene objects: [’table’, ’cyan box’, ’hook’, ’blue box’, ’rack’, ’red box’]
Object relationships: [’on(hook, table)’, ’on(rack, table)’, ’on(blue box, table)’, ’on(cyan box, table)’,
’on(red box, table)’]
Human instruction: How would you push all the boxes under the rack?
Goal predicate set: [[’under(blue box, rack)’, ’under(cyan box, rack)’, ’under(red box, rack)’]]
Top 1 robot action sequences: [’pick(blue box)’, ’place(blue box, table)’, ’pick(hook)’, ’push(cyan box, hook,
rack)’, ’place(hook, table)’, ’pick(blue box)’, ’place(blue box, table)’, ’pick(hook)’, ’push(blue box, hook,
rack)’, ’push(red box, hook, rack)’]
```

```
Available scene objects: ['table', 'cyan box', 'red box', 'hook', 'rack']
Object relationships: ['on(hook, table)', 'on(rack, table)', 'on(cyan box, rack)', 'on(red box, rack)']
Human instruction: put the hook on the rack and stack the cyan box above the rack - thanks
Goal predicate set: [['on(hook, rack)', 'on(cyan box, rack)']]
Top 1 robot action sequences: ['pick(hook)', 'pull(cyan box, hook)', 'place(hook, rack)', 'pick(cyan box)',
'place(cyan box, rack)']
```

```
Available scene objects: ['table', 'rack', 'hook', 'cyan box', 'yellow box', 'red box']
Object relationships: ['on(yellow box, table)', 'on(rack, table)', 'on(cyan box, table)', 'on(hook, table)',
'on(red box, rack)']
Human instruction: Pick up any box.
Goal predicate set: [['inhand(yellow box)'], ['inhand(cyan box)']]
Top 1 robot action sequences: ['pick(yellow box)']
```

```
Available scene objects: ['table', 'blue box', 'cyan box', 'hook', 'rack', 'red box', 'yellow box']
Object relationships: ['inhand(hook)', 'on(red box, rack)', 'on(yellow box, table)', 'on(blue box, table)',
'on(cyan box, rack)', 'on(rack, table)']
Human instruction: could you move all the boxes onto the rack?
Goal predicate set: [['on(yellow box, rack)', 'on(blue box, rack)']]
Top 1 robot action sequences: ['pull(yellow box, hook)', 'place(hook, table)', 'pick(yellow box)', 'place(yellow
box, rack)', 'pick(blue box)', 'place(blue box, rack)']
```

```
Available scene objects: ['table', 'blue box', 'red box', 'hook', 'rack', 'yellow box']
Object relationships: ['on(hook, table)', 'on(blue box, table)', 'on(rack, table)', 'on(red box, table)',
'on(yellow box, table)']
Human instruction: situate an odd number greater than 1 of the boxes above the rack
Goal predicate set: [['on(blue box, rack)', 'on(red box, rack)', 'on(yellow box, rack)']]
Top 1 robot action sequences: ['pick(hook)', 'pull(blue box, hook)', 'place(hook, table)', 'pick(blue box)',
'place(blue box, rack)', 'pick(red box)', 'place(red box, rack)', 'pick(yellow box)', 'place(yellow box, rack)']
```

```
Available scene objects: ['table', 'cyan box', 'hook', 'red box', 'yellow box', 'rack', 'blue box']
Object relationships: ['on(hook, table)', 'on(red box, table)', 'on(blue box, table)', 'on(cyan box, table)',
'on(rack, table)', 'under(yellow box, rack)']
Human instruction: How would you get the cyan box under the rack and then ensure the hook is on the table?
Goal predicate set: [['under(cyan box, rack)', 'on(hook, table)']]
Top 1 robot action sequences: ['pick(blue box)', 'place(blue box, table)', 'pick(red box)', 'place(red box,
table)', 'pick(hook)', 'push(cyan box, hook, rack)', 'place(hook, table)']
```

```
Available scene objects: ['table', 'cyan box', 'hook', 'yellow box', 'blue box', 'rack']
Object relationships: ['on(hook, table)', 'on(yellow box, rack)', 'on(rack, table)', 'on(cyan box, rack)']
Human instruction: set the hook on the rack and stack the yellow box onto the table and set the cyan box on the
rack
Goal predicate set: [['on(hook, rack)', 'on(yellow box, table)', 'on(cyan box, rack)']]
Top 1 robot action sequences: ['pick(yellow box)', 'place(yellow box, table)', 'pick(hook)', 'pull(yellow box,
hook)', 'place(hook, table)']
```

```
Available scene objects: ['table', 'cyan box', 'hook', 'rack', 'red box', 'blue box']
Object relationships: ['on(hook, table)', 'on(blue box, rack)', 'on(cyan box, table)', 'on(red box, table)',
'on(rack, table)']
Human instruction: Move the warm colored box to be underneath the rack.
Goal predicate set: [['under(red box, rack)']]
Top 1 robot action sequences: ['pick(blue box)', 'place(blue box, table)', 'pick(red box)', 'place(red box,
table)', 'pick(hook)', 'push(red box, hook, rack)']
```

```
Available scene objects: ['table', 'blue box', 'hook', 'rack', 'red box', 'yellow box']
Object relationships: ['on(hook, table)', 'on(red box, table)', 'on(blue box, table)', 'on(yellow box, rack)',
'on(rack, table)']
Human instruction: Move the ocean colored box to be under the rack and ensure the hook ends up on the table.
Goal predicate set: [['under(blue box, rack)']]
Top 1 robot action sequences: ['pick(red box)', 'place(red box, table)', 'pick(yellow box)', 'place(yellow box,
rack)', 'pick(hook)', 'push(blue box, hook, rack)', 'place(hook, table)']
```

```
Available scene objects: ['table', 'cyan box', 'hook', 'rack', 'red box', 'blue box']
Object relationships: ['on(hook, table)', 'on(cyan box, rack)', 'on(rack, table)', 'on(red box, table)',
'inhand(blue box)']
Human instruction: How would you set the red box to be the only box on the rack?
Goal predicate set: [['on(red box, rack)', 'on(blue box, table)', 'on(cyan box, table)']]
Top 1 robot action sequences: ['place(blue box, table)', 'pick(hook)', 'pull(red box, hook)', 'place(hook,
table)', 'pick(red box)', 'place(red box, rack)', 'pick(cyan box)', 'place(cyan box, table)']
```

Appendix C
Real world demonstration

1) Hardware setup: We use a kinect v2 camera for RGB-D images and manually adjust the color thresholds to segment the objects. Our segmentations allow us to estimate the object poses using the depth image, which we use to construct environment geometric states. We run robot experiments on a Franka robot arm.

2) Robot demonstration: Please see our project page for demonstrations of Text2Motion operating on a real robot.

Appendix D
Frequently asked questions

Q1: *How generalizable is the framework to include skills such as navigation?*

As with any framework that relies on skill libraries, Text2Motion is ultimately limited by the skills in the skill library. Skills in our framework are trained independent of any long-horizon task. Thus, skills can be added and retrained to support generalization to novel scenarios. A navigation skill can be included in our framework if associated with a parameterized navigation primitive (e.g. RRT* [10]), assuming that object locations in the environment are known and transitions are deterministic. That said, the framework does assume that environment dynamics are deterministic and this assumption may not hold if, for example, we do not know the poses of all objects in the environment *a priori*.

Q2: *Why don't you use use a beam size instead of a greedy search during Text2Motion's integrated search procedure?*

We have experimented with using a beam search with beam size greater than one for Text2Motion's integrated search. However, we found that the LLM was unable to produce calibrated likelihood scores of differing skill sequences relative to each other. For example, the score of a skill sequence that leads to instruction satisfication might have a lower score than one that did not lead to instruction satisfaction. We hope to explore methods [7] to calibrate LLM likelihood scores in future work.