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 skill-level plan that is verified to reach inferred symbolic goals. Text2Motion uses skill feasibility heuristics encoded in learned Q-functions 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 skill sequence optimization 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 64%, 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

Task and Motion Planning (TAMP) refers to a problem setting in which robots solve long-horizon tasks using symbolic and geometric reasoning [9]. Typically, a task planner generates a symbolic plan, while a motion planner optimizes the plan and checks it for geometric feasibility. TAMP methods [9, 13, 16, 22, 6, 3, 8] generally contain three core components: a forward dynamics model and full state observability, meticulously defined symbolic planning domains, and a process to test task plans for geometric feasibility. Several recent works [2, 12, 18, 23] have capitalized on the ability of *Large Language Models* (LLMs) [4] to perform planning for robot systems without needing to manually specify symbolic planning domains. However, challenges remain: how can we verify the correctness and feasibility of LLM-generated plans on the symbolic and geometric level?

We propose Text2Motion, a language-based planning framework that interfaces an LLM with a library of learned skills and a skill sequence optimizer [1] to solve geometrically complex sequential manipulation tasks (Fig. 1). Our contributions are two-fold: (i) an integrated search algorithm which interleaves LLM task planning with skill sequence optimization 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 integrated search method achieves a success rate of 64% 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.

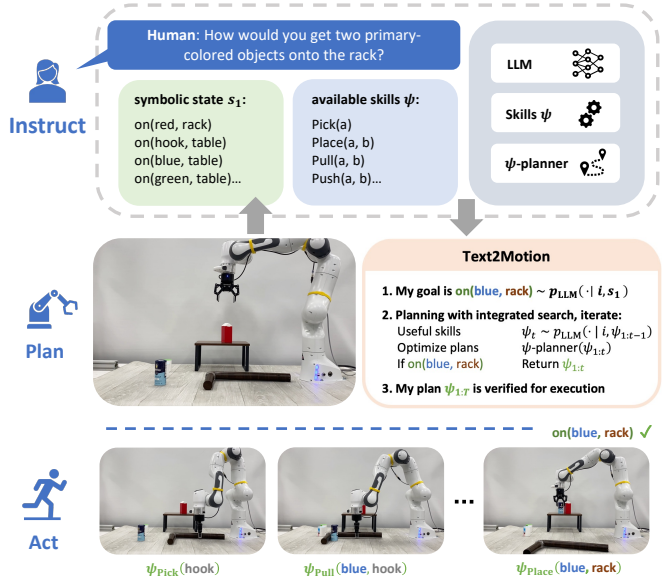

Fig. 1. To carry out the instruction "get two primary-colored objects onto the rack", the robot must apply symbolic reasoning over the scene description and natural language instruction to deduce what skills should be executed to acquire a second primary-colored object, after noticing that a red object 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 [2, 12] that myopically executes skills a the current timestep, Text2Motion constructs plans of skills and coordinates their geometric dependencies through skill sequence optimization [1]. 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. We are given an LLM and a library of skills $\mathcal{L}^{\psi} = \{\psi^1, \dots, \psi^N\}$. Each skill ψ consists of a policy $\pi(a|s)$ and a parameterized manipulation primitive $\phi(a)$ [7], 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 $\pi(a|s)$

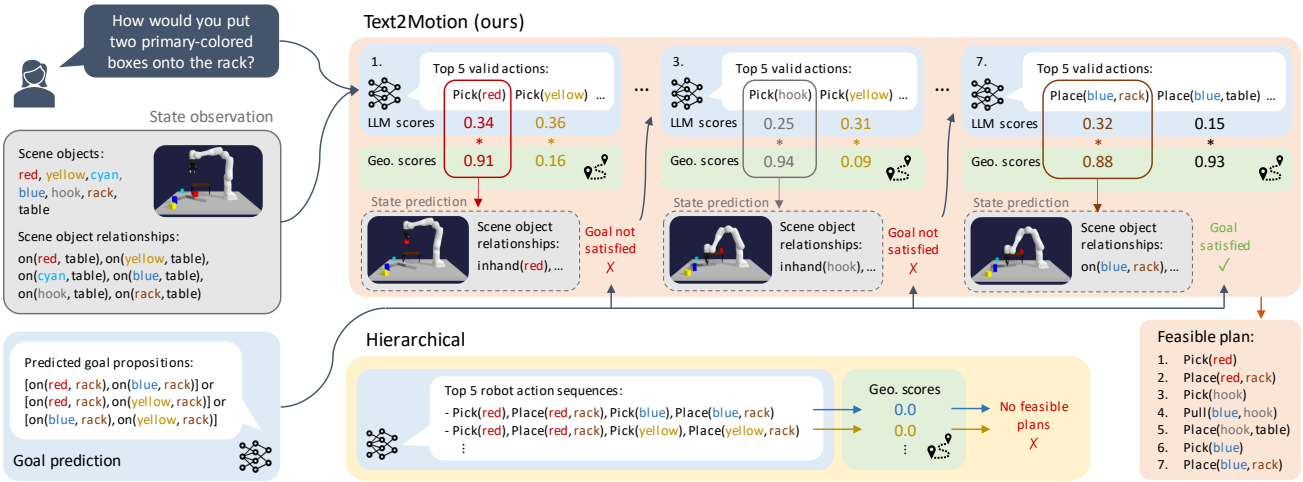

Fig. 2. Text2Motion planning overview. The user provides a natural language instruction for the robot, and then Text2Motion outputs a feasible sequence of skills to solve the given task. First, we use the LLM to predict the set of valid goal state propositions given the instruction and current state (left). These predicted goal states will be used during planning to decide when the goal is satisfied and planning can be terminated. In the first planning iteration, Text2Motion uses the LLM to propose K candidate skills with the top LLM scores. The geometric feasibility planner then performs skill sequence optimization and 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 start the next planning iteration. If the goal propositions are satisfied, then the planner returns. The HIERARCHICAL method, which is used as an experiment baseline, uses the LLM to propose entire plans first and then runs skill sequence optimization afterwards. As shown in the experiments, this approach fails when the space of candidate task plans is large but few skills are geometrically feasible.

and fed to its primitive $\phi(a)$ which consumes the action and executes a series of motor commands on the robot. We subsequently refer to policy actions $a \in \mathcal{A}$ as *parameters* for the primitive, which can represent grasp poses, placement locations, pulling or pushing distances, depending on the skill (Appx. A-A for details). The environment is one in which each timestep corresponds to the execution of a single skill.

We assume that each skill comes with a language description and that methods exist to learn its policy $\pi(a|s)$, action-value function (Q-function) $Q^\pi(s,a)$, and dynamics model $T^\pi(s'|s,a)$. Our method is agnostic to the approach used to learn these models. We also assume a method to convey the environment state $s \in \mathcal{S}$ to the LLM as natural language.

The task planning objective is to find a sequence of skills $[\psi_1, \ldots, \psi_H]$ that is likely to satisfy the instruction i (we hereafter represent sequences with range subscripts, e.g. $\psi_{1:H}$), equivalent to maximizing the language model likelihood of the skill sequence $\psi_{1:H}$ given instruction i and initial state s_1:

$$p(\psi_{1:H} \mid i, s_1). \tag{2}$$

However, we must also consider the geometric feasibility of the skill sequence, which is determined by the parameters $a_{1:H}$ of the underlying primitive sequence $\phi_{1:H}$. A geometrically feasible plan is one where each skill ψ_h receives a reward r_h given the parameter a_h; if just one skill fails (reward $\neg r_h$), then the entire plan fails. The geometric feasibility is defined to be the probability that all skills $\psi_{1:H}$ achieve rewards $r_{1:H}$:

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

Taking the product of Eqs. 2 and 3 results in an objective which represents the probability that a skill sequence $\psi_{1:H}$ is both likely to satisfy instruction i and is geometrically feasible:

$$\begin{aligned} &p(\psi_{1:H}, r_{1:H} \mid i, s_1) \\ &\quad = p(\psi_{1:H} \mid i, s_1)\, p(r_{1:H} \mid i, s_1, \psi_{1:H}). \end{aligned} \tag{4}$$

III. TEXT2MOTION

Our goal is to find a geometrically feasible plan for a given natural language instruction. We run an iterative search, where at each iteration we query the LLM for candidate skills, select the skill with the highest combined language model likelihood and geometric feasibility score, and evaluate whether the search can be terminated. We provide a visualization in Fig. 2.

Our iterative approach can be described as a factorization of the joint probability in Eq. 4 by timestep h:

$$p(\psi_{1:H}, r_{1:H} \mid i, s_1) \approx \prod_{h=1}^{H} p(\psi_h, r_h \mid i, s_1, \psi_{1:h-1}). \tag{5}$$

In Eq. 5, we assume that skill ψ_h and reward r_h can be determined solely from initial state s_1 and sequence of prior skills $\psi_{1:h-1}$, and thus is independent of prior rewards $r_{1:h-1}$.

We define the joint probability of ψ_h and r_h in Eq. 5 as the skill score S_{skill} and factorize it into conditional probabilities:

$$\begin{aligned} S_{\text{skill}}(\psi_h) &= p(\psi_h, r_h \mid i, s_1, \psi_{1:h-1}) \\ &= p(\psi_h \mid i, s_1, \psi_{1:h-1})\, p(r_h \mid i, s_1, \psi_{1:h}) \end{aligned} \tag{6}$$

Each planning iteration is responsible for finding the skill ψ_h that maximizes the skill score S_{skill} at timestep h.

The first factor in this product (Eq. 6) is defined to be the language model likelihood score:

$$S_{\text{llm}}(\psi_h) = p(\psi_h \mid i, s_1, \psi_{1:h-1}). \tag{7}$$

This represents the task planning step, where at timestep h, we ask the LLM to predict skill ψ_h given instruction i, initial state s_1, and sequence of previously planned skills $\psi_{1:h-1}$. The second factor is defined as the geometric feasibility score:

$$S_{\text{geo}}(\psi_h) = p(r_h \mid s_1, \psi_{1:h}), \tag{8}$$

where the reward r_h is conditionally independent of instruction i given the initial state s_1 and skill sequence $\psi_{1:h}$. This represents the motion planning step, where we optimize the skill sequence $\psi_{1:h}$ and evaluate its geometric feasibility.

The skill score to be optimized at each iteration of planning is therefore the product of the language model likelihood and the geometric feasibility of the planned skill sequence:

$$S_{\text{skill}}(\psi_h) = S_{\text{llm}}(\psi_h) \cdot S_{\text{geo}}(\psi_h). \tag{9}$$

Text2Motion alternates between incrementally optimizing $S_{\text{llm}}(\psi_h)$ and $S_{\text{geo}}(\psi_h)$ at each timestep h. The computation of these scores as well as the terminating strategy for the algorithm are described in more detail in the following subsections.

A. LLM task planning

At each iteration h, we optimize $S_{\text{llm}}(\psi_h)$ (Eq. 7) by querying an LLM to generate K candidate skills $\{\psi_h^1, \ldots, \psi_h^K\}$, each accompanied by their own language model likelihood score $S_{\text{llm}}(\psi_h^k)$. These scores represent the likelihood of skill ψ_h^k being the correct skill to execute from a language modeling perspective to satisfy instruction i. These candidates are then checked for geometric feasibility as described in Sec. III-B.

The input prompt contains context c in the form of several planning examples to inform the LLM of the available skills in the robot's library and their usage semantics. The prompt also contains the instruction i, history of planned skills $\psi_{1:h-1}$, initial state s_1, and predicted states $s_{2:h}$ up to step h that describe spatial relationships between objects in the form of symbolic facts (grounded predicates, e.g. on(milk, table)). We refer to Appx. B-B for further prompt details.

B. Geometric feasibility planning

A skill predicted by an LLM may either be symbolically incorrect (e.g. grasp an object while already holding something) or may neglect geometric constraints (e.g. robot kinematics) in the current state. Thus, we propose to compute the geometric feasibility score $S_{\text{geo}}(\psi_h)$ (Eq. 8) of skill ψ_h after maximizing the combined success probability of the entire skill sequence $\psi_{1:h}$. We term this process *skill sequence optimization*. As a result, the skills $\psi_{1:h-1}$ are expected to receive rewards $r_{1:h-1}$, which supports the independence assumption of Eq. 5.

We resolve geometric dependencies across the full skill sequence $\psi_{1:h}$ by maximizing the product of step reward probabilities [1] given parameters $a_{1:h}$ of primitives $\phi_{1:h}$:

$$a_{1:h}^* = \arg\max_{a_{1:h}} \prod_{t=1}^{h} p(r_t \mid s_t, a_t), \tag{10}$$

where future states $s_{2:h}$ are predicted by dynamics models $s_{t+1} \sim T^{\pi_t}(\cdot|s_t, a_t)$. Note that the reward probability $p(r_t \mid s_t, a_t)$ is equivalent to the Q-value $Q^{\pi_t}(s_t, a_t)$ for skill ψ_t in a contextual bandit setting with binary rewards (Eq. 1). We thus estimate the geometric feasibility $S_{\text{geo}}(\psi_h)$ (Eq. 8) by the Q-value of the optimized parameter a_h^* computed from Eq. 10,

$$p(r_h \mid s_1, \psi_{1:h}) \approx Q^{\pi_h}(s_h, a_h^*), \tag{11}$$

where s_h is predicted by the dynamics model. This geometric feasibility score (Eq. 11) is multiplied by the language model likelihood (Eq. 7) to produce the skill score (Eq. 9):

$$S_{\text{skill}}(\psi_h) \approx S_{\text{llm}}(\psi_h) \cdot Q^{\pi_h}(s_h, a_h^*). \tag{12}$$

This process is performed for each of the K candidate skills $\{\psi_h^1, \ldots, \psi_h^K\}$. Invalid skills as determined by Sec. III-C are filtered-out of the candidate set. Of the remaining skills, the one with the highest skill score is added to the running plan.

C. Out-of-distribution detection

During planning, the LLM may propose skills ψ_h^k that are out-of-distribution (OOD) given the current state s_h and optimized parameter a_h^*. We consider a skill ψ_h to be OOD if the variance of its associated Q-value (Eq. 11) predicted by an ensemble [17] exceeds a calibrated threshold ϵ^{ψ_h}:

$$F_{\text{OOD}}(\psi_h) = \mathbb{1}\left(\text{Var}_{i\sim 1:B}\left[Q_i^{\pi_h}(s_h, a_h^*)\right] \geq \epsilon^{\psi_h}\right),$$

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 for ensembles of Q-functions.

D. LLM goal prediction

Given a library of predicate classifiers $\mathcal{L}^\chi = \{\chi^1, \ldots, \chi^M\}$, a set of objects O in the scene, and an instruction i, we use the LLM to predict a set of symbolic goal states $\mathcal{G} = \{g_1, \ldots, g_j\}$ 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 (e.g. on(a, b)). We define a satisfaction function $F_{\text{sat}}^{\mathcal{G}}(s) : \mathcal{S} \to \{0, 1\}$ that converts the geometric state s into a symbolic state using predicate classifiers $\mathcal{L}^\chi$ and checks if any goal $g \in \mathcal{G}$ is satisfied. $F_{\text{sat}}^{\mathcal{G}}$ is used to: a) terminate planning if the predicted geometric state s_{h+1} resulting from the selected skill ψ_h satisfies the instruction i, and b) construct a chain-of-thought prompt [24] as a sequence of symbolic states when querying the LLM for candidate skills (as described in Sec. III-A).

IV. Experiments

We conduct experiments to test the following hypothesis: (**H1**) skill sequence optimization is necessary for solving TAMP-like problems specified by natural language with LLMs and robot skills; (**H2**) integrated reasoning is required to solve tasks with partial affordance perception (defined in Fig 3).

A. Baselines

We compare Text2Motion with three language-based planners and use the same skill library for all methods and tasks.

SAYCAN-GS: A cost-considerate variant of SayCan [2] that queries the LLM for the K most useful skills (Generator) before scoring them (Scorer). The skill with the highest combined value function and language model score is executed.

INNERMONO-GS: The *Object + Scene* variant of Inner Monologue [12] that receives task-progress scene context in the form of the environment's symbolic state. We obtain INNERMONO-GS by equipping [12] with the Generator-Scorer.

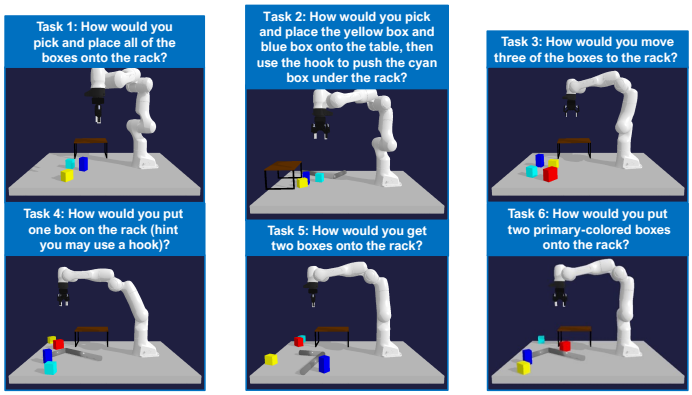

Fig. 3. **TableEnv Manipulation evaluation task suite**. We evaluate the performance of all methods on a task suite based on the above manipulation domain. The tasks considered vary in terms of difficulty and each contains a subset of three properties: i) (**LH**: Tasks 1, 2, 3, 5, 6) long-horizon tasks 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.

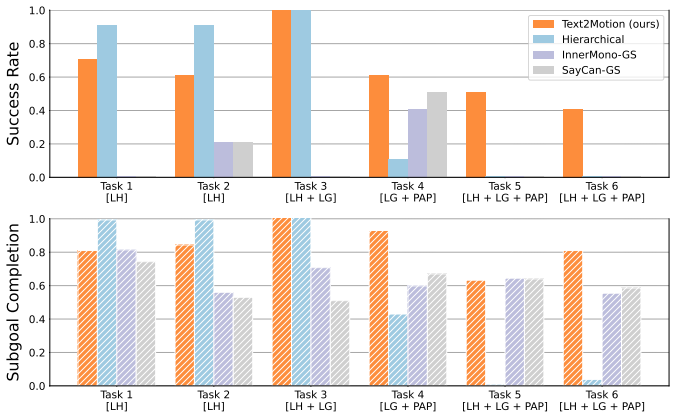

Fig. 4. **Results on the TableEnv manipulation domain** with 10 seeds for each task. **Top:** Our method (Text2Motion) outperforms all baselines on tasks involving partial affordance perception (Task 4, 5, 6). For tasks without partial affordance perception, methods that use skill sequence optimization (ours and HIERARCHICAL) outperform the methods (SAYCAN-GS and INNERMONO-GS) that do not use skill sequence optimization. We note HIERARCHICAL performs well on tasks without partial affordance perception (Tasks 1, 2, 3) as it outputs *multiple* goal-reaching candidate task plans and selects the most geometrically feasible. **Bottom:** Methods without skill sequence optimization 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.

LLM skill likelihoods are equivalent to those from SAYCAN-GS but now also conditioned on the visited state history.

HIERARCHICAL: We propose a baseline method, termed the HIERARCHICAL method. Like Text2Motion, HIERARCHICAL predicts a set of goal propositions and converts them into a satisfaction function $F^{\mathcal{G}}_{\text{sat}}$ (Sec. III-D). Unlike Text2Motion, HIERARCHICAL uses the LLM to generate entire skill sequences $\psi_{1:H}$ (Eq. 2), runs skill sequence optimization for each of the candidate sequences, and filters out plans with low predicted Q-values or OOD state-action pairs. Finally, if multiple plans contain geometric states satisfying $F^{\mathcal{G}}_{\text{sat}}$, we select the plan with the highest success probability (Eq. 10).

B. Large language model

We use two language models, both of which were accessed through the OpenAI API: `text-davinci-003` [20] and Codex [5] (`code-davinci-002`). To generate task plans for the HIERARCHICAL method, we empirically found `text-davinci-003` to be the most capable for HIERARCHICAL's task plan generation queries. For other queries, we use `code-davinci-002`.

C. Prompt engineering

We use the same set of in-context examples for all methods and for all tasks in the prompts passed to the LLM. A single prompt contains a description of the scene objects and object relationships, the instruction, a goal predicate set and the current query, such as "`Top 5 next valid robot actions: `". We provide the full set of in-context examples in Appx. B-B.

D. Evaluation and metrics

We construct a suite of evaluation tasks in a table-top manipulation domain (Fig. 3). Text2Motion and HIERARCHICAL produce a successful plan if, upon execution, the plan reaches a final state s_H that satisfies the instruction i. A plan is executed only if the final state s_H predicted by the dynamics model satisfies $F^{\mathcal{G}}_{\text{sat}}$ (Sec. III-D). Both methods perform closed-loop execution of the skill sequence $\psi_{1:H}$, calling [1] to perform skill sequence optimization on the skills $\psi_{h+1:H}$ after executing skill ψ_h. We do not perform task-level replanning.

SAYCAN-GS and INNERMONO-GS are reactive agents that execute the next best admissible skill ψ_h 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.

V. Results

The results are presented in Fig. 4. We find that skill sequence optimization is indeed required to solve TAMP-like problems with LLMs (**H1**) and that integrated reasoning (exemplified by Text2Motion) is required for PAP tasks (**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 skill sequence optimization, integrated planning, and plan termination via inferred symbolic constraints for solving TAMP-like tasks from a natural language instruction.

References

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

[2] 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.

[3] Julien Bidot, Lars Karlsson, Fabien Lagriffoul, and Alessandro Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 247:229–265, 2017.

[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] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, AnnArbor, Michigan, June 2016. doi: 10.15607/RSS.2016.XII.002.

[7] 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.

[8] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020.

[9] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[10] 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.

[11] 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.

[12] 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.

[13] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391.

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

[15] 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.

[16] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 33(14):1726–1747, 2014.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[23] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft, February 2023. URL https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/.

[24] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain

of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[25] 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 the language planning baselines (Appx. A), the experimental setup (Appx. B), and the **real-world demonstrations** (Appx. C). We provide a discussion of commonly asked questions in Appx. D. Qualitative results are made available at sites.google.com/stanford.edu/text2motion.

Appendix A: Implementation Details 8

A-A Learning robot skills and dynamics 8

A-B Out-of-distribution detection 9

A-C Task planning with LLMs 9

A-D Geometric feasibility planning 10

Appendix B: Experiment Details 11

B-A Scene descriptions as symbolic states 11

B-B In-Context Examples 11

Appendix C: Real world demonstration 14

Appendix D: Frequently asked questions 15

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 HIERARCHICAL 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 $\mathcal{L} = \{\psi^1, \ldots, \psi^N\}$. Each skill ψ has a language description (e.g. Pick(a)) and is associated with a parameterized manipulation primitive [7] $\phi(a)$. A primitive $\phi(a)$ is controllable via its *parameter* a which determines the motion [15] 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 Appx. 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 [11]:

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

noting that maintaining some degree of policy stochasticity is beneficial for skill sequence optimization (Sec. A-D). 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 [19]. 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 [11]. 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 [17]. A state-action pair is classified as OOD if the empirical variance of the predicted Q-values is above a determined threshold:

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

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. Task planning with LLMs

Text2Motion and the reactive planning baselines SAYCAN-GS and INNERMONO-GS uses `code-davinci-002` (Codex model [5]) generate and score skills, while HIERARCHICAL queries `text-davinci-003` (variant of InstructGPT [20]) to directly output full skill sequences. Surprisingly, a temperature of 0 was found to be sufficient for our 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 Text2motion considers five possible extensions to the current running sequence of skills $\psi_{1:h-1}$. Similarly, HIERARCHICAL generates $K = 5$ skill sequences.

As described in Sec. III-A, 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 uses skill sequence optimization 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

Text2Motion relies on a technique to compute the geometric feasibility score of a skill $S_{\text{geo}}(\psi_h) = p(r_h \mid i, s_1, \psi_{1:h})$ in the context of the running plan $\psi_{1:h-1}$. As detailed in Appx. A-C, this probability can be computed without considering geometric action dependencies—particularly if actions have already been executed, as is the case with SAYCAN-GS and INNERMONO-GS. However, Text2Motion computes $S_{\text{geo}}(\psi_h)$ after skill policy sequence optimization; the process by which long-horizon geometric dependencies are coordinated across skill sequences $\psi_{1:h}$. For example, this would ensure that when evaluating the selection of Pull(box, hook) during integrated search, the algorithm does so in the context of a suitable Pick(hook) action (e.g. an upper-handle grasp on hook).

Text2Motion is agnostic the method that fulfils this role. However, in our experiments we leverage Sequencing Task-Agnostic Policies (STAP) [1]. 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) [21], 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 (Sec. 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 [25]. 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* [14]), 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 a beam size larger than 1 during Text2Motion's integrated search procedure?*

We have experimented with using larger beam sizes 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 [10] to calibrate LLM likelihood scores in future work.