2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

GenCHiP: Generating Robot Policy Code for High-Precision and Contact-Rich Manipulation Tasks

Kaylee Burns*[1,2], Ajinkya Jain[1], Keegan Go[1], Fei Xia[3], Michael Stark[1], Stefan Schaal[1], Karol Hausman[2,3]

***Abstract*— Large Language Models (LLMs) have been successful at generating robot policy code, but so far these results have been limited to high-level tasks that do not require precise movement. It is an open question how well such approaches work for tasks that require reasoning over contact forces and working within tight success tolerances. We find that, with the right action space, LLMs are capable of successfully generating policies for a variety of contact-rich and high-precision manipulation tasks, even under noisy conditions, such as perceptual errors or grasping inaccuracies. Specifically, we reparameterize the action space to include compliance with constraints on the interaction forces and stiffnesses involved in reaching a target pose. We validate this approach on subtasks derived from the Functional Manipulation Benchmark (FMB) and NIST Task Board Benchmarks. Exposing this action space alongside methods for estimating object poses improves policy generation with an LLM by greater than 3x and 4x when compared to non-compliant action spaces. More material is available on our project webpage: https://dex-code-gen.github.io/dex-code-gen/**

I. Introduction

Many of the open problems in learning-based robotics revolve around the issue of scaling: deep-learning methods require vast datasets that are not readily available for robotics applications. One workaround for the data scarcity problem is to retrofit deep learning models that have been trained on internet-scale datasets from other modalities for robotics tasks. Recently, large language models (LLMs) have emerged as a strong candidate for this approach. LLMs are able to successfully generate code, complete numeric sequences, and solve common-sense reasoning tasks [1]–[4]. Because code is one of the most popular interfaces for specifying robotic planning and control commands, these capabilities hint at enormous potential when applied to robotics [1].

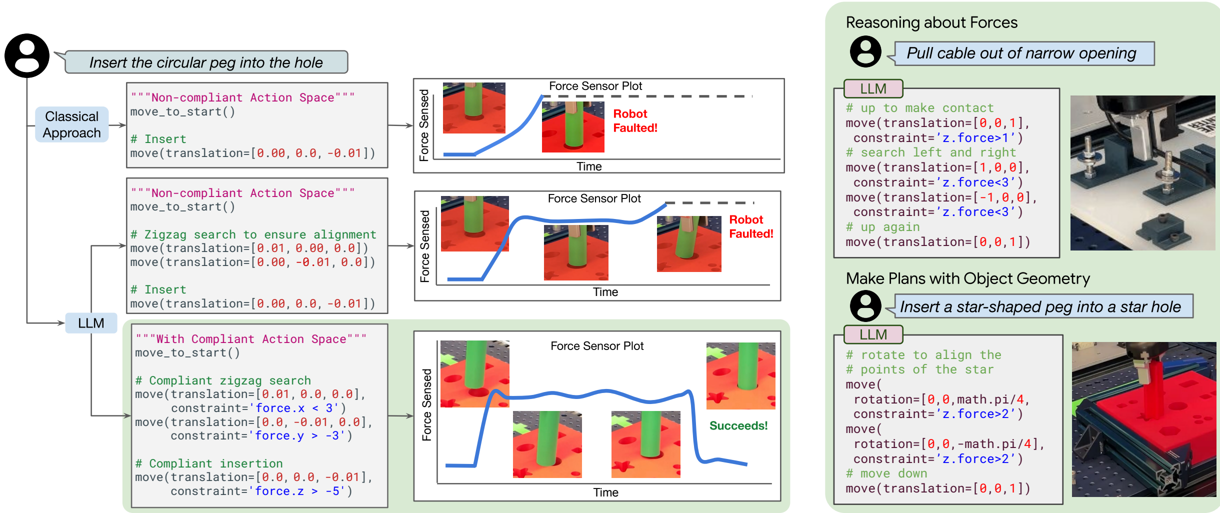
Past work demonstrates that generating robot policy code from LLMs is successful for high-level tasks such as navigation and open-vocabulary pick-and-place [1], [2]. For example, a language model can compose high-level action primitives like `grab(chips)` and `move_to(human)` to successfully generate a policy conditioned on a natural language command such as "bring me the chips" [1]. But at present, lower-level tasks and behaviors are generally considered out of reach for LLMs because, to the best of our knowledge, there are no compelling demonstrations of code generation that commands robots to perform contact-rich tasks.

While various robot learning approaches have been able to demonstrate impressive generalization across different settings and target objects for pick-and-place tasks [5]–[10], such generalization is arguably more difficult for dexterous tasks where a higher level of precision is required. For example, for a peg-in-hole insertion task, surfaces with more friction or tight insertion tolerances may require multiple insertion attempts or contact force tuning to reach the insertion site [11]. Similarly, pegs with different geometries may need different approach trajectories to achieve proper alignment: a peg with a star-shaped cross-section may require an initial rotation for insertion, whereas no such rotation is required for a circular peg. In practice, the parameters of contact-rich insertion skills are mostly tuned by experts to handle these differences and automating this process is still an open problem. This presents a challenge for the approach taken in past work [1], which directly provides a language model with a library of high-level skills (such as `insert(peg)`).

We present GenCHiP: a promising alternative for automating the parameter-tuning process within the control API by instead leveraging the world knowledge inside language models (LMs) to compose lower-level control primitives. Our goal is to understand if LLMs have the ability to reason about motions and forces acting on objects so as to enable generalization across a larger class of target objects and skills. To study this topic, we modify the action space in which an LLM operates by exposing constraints on the contact stiffness and forces observed during a high-precision task. With these modifications we can study code generation for contact-rich manipulation tasks, including industry relevant tasks such as high-precision insertion, rigid body assembly, and deformable object manipulation (see Fig. 1).

The main contribution of this work is to demonstrate that LLMs, without any specialized training, have the ability to perform contact-rich tasks when given the appropriate action space. We develop a system for automatically generating robot policy code for dexterous tasks by allowing LLMs to specify constraints on the stiffnesses, forces, and trajectories required to perform contact-rich manipulation tasks. We show that our method, GenCHiP, is able to outperform a contact-unaware model by over 3x on average on subtasks developed from two challenging contact-rich benchmarks. Specifically, our approach is able to generate novel insertion patterns from high level descriptions of object shape and texture on insertion tasks from the Functional Manipulation Benchmark (FMB) [12], route and un-route cables in the style of the IROS 2020 Robotic Grasping and Manipulation Competition (IROS RGMC) [13], and complete a waterproof connector insertion task selected from the NIST Assembly Task Board #1 [14]. To the best of our knowledge, this is the first paper to provide a strong proof-of-concept that LLMs

*Correspondence to `kayl@google.com`
[1][Google] Intrinsic [2]Stanford University [3]Google DeepMind



(a) (b)

Fig. 1: (a) We prompt an LLM to generate code for high-precision tasks. By using an action space that paramterizes compliant behavior, the LLM is able to generate action sequences for contact-rich tasks like peg insertion. (b) Language models' ability to reason about object geometry and make plans by using world knowledge about different object enables zero-shot generalization to new tasks.

are capable of parameterizing robotic policies for precise, contact-rich manipulation tasks with code.

II. Related Work

LLMs for robotics can successfully generate robot policy code for pick-and-place style manipulation tasks [15], [16], compose mid-level plans for navigation tasks [3], and compose multiple navigation and manipulation skills for integrated household agents [17]–[19]. Many of these approaches rely on filtering LLM-generated code based on what is executable [20] or on making queries hierarchically [1], [10]. They also require a large number of prompt examples: for example, Huang, Wang, Zhang, *et al.* [21] makes use of eight different prompts with approximately nine subcommands per prompt. By contrast, in the Few-Shot example settings we describe in Sec. IV-A, we use a maximum of four examples in a given environment. Furthermore, none of these works establish whether or not LLMs can generate robot policy code for performing high-precision, contact-rich manipulation tasks, which we study in this work.

Contact-rich robot manipulation tasks are those that involve a robot making controlled contact with its environment while performing them. These tasks constitute the vast majority of manipulation tasks in daily life, including household tasks such as wiping tables and sweeping dust into a dust-pan [22], and industrial tasks such as high precision insertion [11], [23], [24] and assembly [25], [26]. A robot needs to reason about the contact forces it will impart on and sense from the environment while performing such tasks to complete them successfully. Learning a general policy to perform a wide array of contact-rich manipulation tasks has been studied in great detail in robotics [23], [24], [27]–[32], yet how to find a general approach to these tasks remains an open question. Prior work directly learns policies with imitation learning [33] or reinforcement learning [25], [34], [35], but these require hundreds of human demonstrations, significant operator training, dedicated simulators, or thousands of environment interactions to achieve a performant policy. We step towards obtaining a general policy for contact-rich manipulation tasks by leveraging the world knowledge inside LLMs and combining it with the appropriate task action spaces. We choose robot impedance (or equivalently, admittance) control as the action space for contact-rich manipulation tasks as it can regulate the relationship between robot position and contact forces effectively [36], [37].

```
# You're a robot trying to insert a peg in a hole. Grab
  the circular peg.
pick_up(circular_peg)
```

Fig. 2: We generate code by formatting natural language requests and instructions as comments. Generations are highlighted in blue.

III. Preliminaries

Our goal is to develop a system that can translate natural language instructions into robotic actions by leveraging a sufficiently expressive API for control. Past work [1] shows that off-the-shelf language models can be adapted towards this goal with few-shot prompting. Concretely, pairs of natural language requests with corresponding robot policy code are fed into a language model. Then, the language model can output novel programs in response to new commands as shown in Fig. 2. The success of this approach can be attributed to the fact that during offline training on vast internet datasets, language models absorb world knowledge about common-sense interactions and learn mappings between natural language instructions and code. Strategies for

```
"""You're a robot trying to undo cable routing. Unroute
   the cable from the screws and brackets it is wrapped
   around."""
```

(a) Task Description

```
"""Use these methods:
- move: moves to specified offset
  Args:
      translation: (x, y, z) tuple
      rotation: (x, y, z) tuple
..."""
```

(b) Control API Descriptions

```
"""Rules:
- Don't define any new methods
- Don't call any undefined methods
- Don't add any if statements or while loops"""
```

(c) Hints

```
# Move the cable to the left until it snags
move((1, 0, 0),
     constraint=(x.force>-1))
```

(d) Examples

Fig. 3: We present information about the task and control API via prompting. The API description is the same across all environments, the hints and examples are the same within each environment, while the task description must be modified to describe each task.

```
# Insert a peg into a hole
pick_up(peg)
# go down to make contact
move_point_to_point([0, 0, -1])
# wiggle to find opening
move_point_to_point([1, 0, 0])
move_point_to_point([-1, 0, 0])
# go down to insert
move_point_to_point([0, 0, -1])
```

Fig. 4: Generating code with point-to-point moves limits policies to free-space-motions. For this policy to run successfully, displacement along the z-axis in the second and fifth actions must be exact.

adapting this approach towards a contact-rich setting are discussed in the next section.

IV. GenCHiP

GenCHiP equips LLMs with a compliant action space. In this section, we first describe the prompting strategies that enable a language model to successfully paramterize and compose compliant move actions. Then, we discuss the different choices of action spaces that can be made available to a language model including our proposed action space.

A. Prompting for contact-rich control

We consider four prompting strategies when generating robot policy code from a language model:

a. **Task descriptions** are high-level explanations of the scene and the task goal written in natural language. These can occur at both the beginning and end of a prompt and often include important information about the task setup such as the peg shape or the available objects. See Fig. 3a for an example.

b. **Descriptions of available control APIs** are formatted doc-strings that describe code accessible to the LLM. These include lists of variables as well as the expected range of values for floating point numbers. Fig. 3b shows an example description for a simplified `move` function that controls end-effector pose. We also include descriptions for the full library of available methods, including a point-to-point move, a compliant move, conditions, gripper movements, and methods or variables specifying the positions of relevant objects.

c. **Hints** in our setting include rules, keywords that specify relevant control primitives, and requests to have the model explain its reasoning in natural language or in pseudocode. Phrases such as "perform a pattern search" guide the model towards predicting behavior that better recovers from errors and better handles imprecision in the position of target poses. Intuitively these keywords help in reducing task ambiguity (e.g., by emphasizing that provided locations are imprecise) and guide the model towards motion patterns that are relevant to contact-rich tasks. Requests to explain in natural language can be thought of as a variant of chain-of-thought prompting [38]. The specific keywords and requests that are helpful in each task are described in the experimental section.

d. **Examples** of the control APIs being used for basic movements, such as making contact with a surface, are useful for tasks with ambiguity or where the desired force constraints are difficult to infer from the given ranges.

The combination of the prompt strategies described above allows us to prompt a language model with enough contextual information about the dexterous task at hand. Next, we discuss how we can design the action space of the robot to be able to perform such tasks in practice.

B. Action spaces for robot manipulation tasks

Past approaches assume access to a library of methods that exhaustively cover all user-requested commands [1], [2]. Building such a library is challenging for contact-rich tasks because in practice these policies are tuned by experts across different object geometries, frictions, and scene layouts. This section describes different approaches to parameterizing the control API and what the right choice of a control API can achieve. Formally, we consider a contact-rich robot manipulation task to be composed of a sequence of subtasks $\tau = (t_1, \dots, t_n)$. The specific definition of a subtask will change based on the action space, as described below.

Point-to-point moves. Past work [1], [2] makes use of an action space that directly commands the robot to move to target poses in the Cartesian space, $[\mathbf{x}_{target}]_i$ (See Fig. 4). In this setting, each sub-task t_i is simply defined as the next Cartesian pose (a.k.a. waypoint): $t_i = ([\mathbf{x}_{target}]_i)$.

While this approach is successful for executing motions in free-space or for simple pick-and-place tasks, it fails when the robot needs to explicitly make a purposeful contact with

```
# Insert a peg into a hole
pick_up(peg)
# go down to make contact
move_conditional_compliant([0, 0, -1],
     constraint=(z.force>1))
# wiggle to find opening; stop when force lessens
move_conditional_compliant([1, 0, 0],
    constraint=(z.force<1))
move_conditional_compliant([-1, 0, 0],
    constraint=(z.force<1))
# go down to insert
move_conditional_compliant([0, 0, -1],
    constraint=(z.force>2))
```

Fig. 5: Compliance prevents the robot from faulting when in contact. Conditional termination constraints enable the language model to reason about contact forces. In this example, the robot moves a cable back and forth until no more upward force is detected, which indicates that an opening has been found.

its environment. Consider a robot trying to make contact with a surface to perform a wiping motion. Successfully parameterizing a policy in this action space would require predicting a precise Cartesian pose with very little tolerance for error. Predicting a pose that is millimeters short of the surface would fail to make a contact and predicting millimeters too deep into the surface would cause the robot to exert high forces on the surface, which can cause faults in the robot or even break it in the worst case scenario.

Compliant moves. Addressing this shortcoming, we propose to parameterize the action space for performing contact-rich manipulation tasks using a robot's compliance, realized in impedance control (or equivalently admittance control which is an adequate action space for robot learning in Martín-Martín, Lee, Gardner, *et al.* [39]). An impedance move action is parameterized by both a target Cartesian pose, $[\mathbf{x}_{target}]_i$, and a vector that specifies stiffness along each degree of freedom, σ_i: $t_i = ([\mathbf{x}_{target}]_i, \sigma_i)$, when the robot is in contact with the environment. During execution, the stiffness vector for each subtask can be used to define the parameters for a variable impedance controller [40] of the form:

$$F_{external} = K_p(\mathbf{x}_{target} - \mathbf{x}) + K_d(\dot{\mathbf{x}}_{target} - \dot{\mathbf{x}}) + \Lambda(\ddot{\mathbf{x}}_{target} - \ddot{\mathbf{x}}) \tag{1}$$

where $\mathbf{x}_{target}$, $\mathbf{x}$, $\dot{\mathbf{x}}_{target}$, $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}_{target}$, and $\ddot{\mathbf{x}}$ denote the target and current pose, twist, and accelerations, respectively. K_p, K_d, and Λ correspond to the stiffness, damping, and task-space inertia matrices, respectively. The impedance controller realizes that the end-effector in contact with the environment behaves like the linear spring-damper-mass system above. K_p, K_d, and Λ are computed as a function of our specified stiffness vector σ_i (explained below) and robot specific parameters in order to achieve stable yet responsive behavior.

Intuitively, the stiffness vector determines the interaction forces that the robot will impart on its environment while performing the task. Low stiffness coefficients in σ regulate the robot's compromise between contact forces and the attempt to achieve position accuracy. In the example that we discussed in the last paragraph, a low stiffness value would enable the robot to maintain gentle contact with a surface that prevents the robot from reaching a desired position. A higher stiffness value would create higher contact forces, equivalent to a higher priority to reduce position error.

Conditional compliant moves. In addition to the impedance control specification described above, we also allow the LLM to specify conditions under which to terminate an impedance move. Specifically, these are thresholds on force or position in a specified coordinate direction. Example pseudocode is presented in Fig. 5. This is a powerful primitive as it enables the robot to construct recipes for high-precision tasks without relying on fine-grained perception. In the example of making contact with a surface, this may look like moving a peg downwards with a termination constraint on upward force.

V. Experiments

In this section, we evaluate the ability of GenCHiP to generate code for fine-grained manipulation tasks that require high precision in a series of experiments on a set of real robotic tasks. Specifically, we evaluate on a subset of high-precision contact-rich manipulation tasks from the Functional Manipulation Benchmark (FMB) [12] and a set of industrial manipulation tasks selected from the NIST Assembly Task Board #1 [14] and the IROS 2020 Robotic Grasping and Manipulation Competition (IROS RGMC) [13] (Sec. V-A). Later, we ablate prompt hints to study the utility of incorporating additional hints in generating relevant motion patterns for robot manipulation tasks (Sec. **??**).

A. Contact-rich manipulation tasks

Sec. V-A.1 details the task setup considered in our experiments, while baselines and ablations are discussed in Sec. V-A.3. We discuss experimental results in Sec. V-A.5.

1) Task description: The **Functional Manipulation Benchmark** [12] studies robotic manipulation, grasping, reorienting, and assembling of a set of dozens of 3D printed objects. The benchmark emphasizes generalization across different object shapes and positions. We evaluate our approach on a subset of peg insertion tasks across three different object shapes: the circle, star, and half-pipe. We use scripted motion to bring the pegs into a fixed position over the insertion point and program a randomized rotation about the z-axis. There is no rotation for the circular peg because it has a constant radius. Rotation of the star is sampled uniformly between 0 and $\frac{\pi}{2}$. Rotation of the half pipe is sampled uniformly from either 0 or π. Inserting these peg shapes successfully requires generating search patterns contingent on object geometry.

Industrial Manipulation Tasks are adapted from the NIST Assembly Task Board #1 [14] and the IROS RGMC 2020 [13]. These benchmarks are designed to evaluate proficiency in robotic assembly with an emphasis on small and medium sized parts and deformable objects. We consider the wire routing and connector insertion subtasks. Specifically, we study routing (insertion) and unrouting

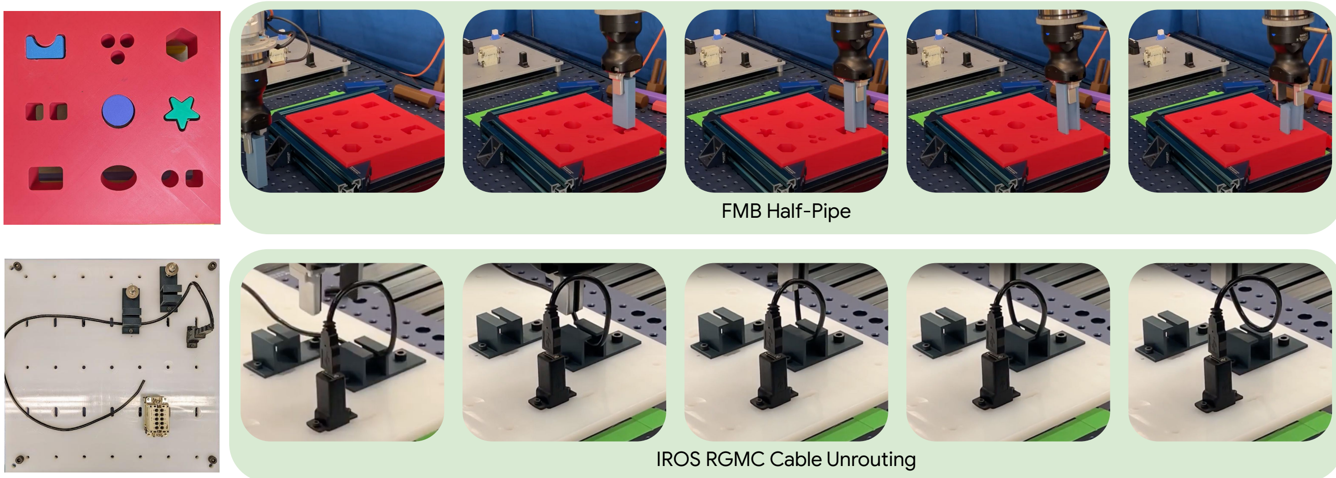


Fig. 6: Left: The Functional Manipulation Benchmark (FMB) [12] and a modified assembly board with tasks sourced from NIST Assembly Task Board #1 [14] and IROS RGMCS 2020 [13] we used for experimentation. Both environments have relatively tight tolerances. Right: Example of the rollouts produced by our method for two tasks: the waypoints from GenCHiP are able to successfully complete the tasks.

(removal) of a wire through a plastic channel component and inserting a waterproof connector plug into a socket. Task environments are visualized in Fig. 6. Note there are multiple sources of noise across episodes. For example, in the orientation of the cable within the grasp of the gripper, the tautness of the cable, and the estimated poses of the waterproof connector. Additionally, we evaluate on each side of the routing component to verify that the pattern works across starting positions. An ideal method should result in an execution policy that is robust to such sources of noise.

2) Implementation details: We conduct our robot experiments on a Universal Robotics UR5e robot, which is a position-controlled robot with ATI Axia80 force-torque sensor at the wrist. To expose the compliant action space to the language model, we prompt it with the doc-string for a Cartesian admittance move with parameters on stiffnesses, impedances, and constraints in reaching a target pose. We add a suffix describing the details of the given task, optionally including certain keywords about relevant motion patterns when the environment setup is ambiguous (e.g., we specify that the peg in the FMB insertion tasks is not aligned, which requires the policy to search for the opening).

We give the LLM access to methods for computing transformations on reference poses (i.e., `pose_multiply`) and for detecting objects in the environment (i.e., `estimate_and_update_pose`). To support the latter, we train pose estimators to do joint object detection and keypoint prediction using a Faster-RCNN [41] architecture.

3) Methods considered: We compare two classes of methods: a scripted baseline policy authored by an expert and different variants of LLM-generated code using the prompting strategies and control APIs outlined in Sec. IV-A and Sec. IV-B. For LLM-generated code, we compare with and without code examples. For zero-shot settings, the prompt includes includes Task Descriptions (Fig. 3a), Control API Descriptions (Fig. 3b), and Hints (Fig. 3c). In few-shot settings, Examples (Fig. 3d) are added to the prompt.

Scripted [Baseline]. We compare against a scripted pattern search insertion move that is tuned by an expert on a single task setting. This baseline reflects an alternative to our approach where a single skill is added to our control library, but is not able to be tuned by an expert across different task generalizations. On FMB tasks, we adapt a pattern search insertion skill for peg insertion. The scripted move implements fixed get-in-contact, pattern search, and insertion phases, with durations, motion patterns, and force thresholds set by an expert on the circle setting.

Code-as-Policies (CaP) [1] [Baseline]. We compare with a baseline approach akin to the prior work [1], [2] that uses the point-to-point action space for performing robot manipulation tasks, i.e. to directly command the robot to move to Cartesian target poses (Fig. 4).

GenCHiP, Fixed Compliance (FC). For each task, we compare against a baseline where we do not expose the stiffness and impedance targets or the force constraints to the LLM planner, but instead, use predefined compliance parameters. This ablates the importance of force constraints in completing the task, making the action space similar to prior work [1], but with compliant motions. Concretely, we provide a modified prompt and access to a wrapper around the Cartesian admittance move that provides fixed stiffness and impedance targets and a fixed translation error constraint.

GenCHiP (Few-Shot). We expose force constraints to the language model and add examples of calls to our control API, which includes conditional compliant moves. This is similar to the GenCHiP, Fixed Compliance baselines, but all of the force constraints and termination conditions are exposed to the language model (Fig. 5). In these experiments, each

	Circle	Star	Half-Pipe
Scripted	**100%**	10%	0%
CaP [1] (Zero-Shot)	70%	0%	0%
GenCHiP, FC (Zero-Shot)	**100%**	70%	30%
GenCHiP (Zero-Shot)	**100%**	**80%**	**50%**

TABLE I: **Functional Manipulation Benchmark. [12]** Compared to a scripted policy, GenCHiP is better able to generalize an insertion pattern across object geometries, including the challenging half-pipe task. We show average performance across 10 evaluations.

"shot" is an example subcommand that shows how to call the API. In the IROS RGMCS tasks, we include 3 example subcommands: moving down until contact, moving up until a snag, and moving right until a snag is detected.

GenCHiP (Zero-Shot). We follow the same approach of exposing force constraints to the language model as GenCHiP (Few-Shot, Fig. 5), but do not include any examples that call the control APIs. This is the most difficult generalization setting because every command is an unseen command.

4) Evaluation Protocol: Similar to Yu, Gileadi, Fu, *et al.* [42], we take the best completion out of 5 calls to the underlying language model and run 10 evaluations. All of our experiments use the OpenAI ChatCompletions API [43] with the `gpt-4-0613` endpoint and a temperature of 0.0. To make comparisons between different action spaces as fair as possible, we take the most successful code generated from our method and overwrite the control API to implement the relevant action space. For each environment, we tune the insertion reference pose that appears in the prompt. Concretely, this is the reference pose used in to make contact in the admittance move. This hyperparameter is essential on the Point-to-Point baseline because insertion reference poses that are too deep cause a fault.

5) Results: **Functional Manipulation Benchmark.** Our first evaluation studies how well different approaches generalize across unseen task settings, in particular, the ability to modify insertion search patterns based on different object geometries. Results on FMB are listed in Tab. I.

We find that GenCHiP outperforms the performance of other methods across different peg shapes. The baseline scripted policy is successful on the star only when the points are already in close-enough alignment with the hole and fails on the half-pipe shape, which is the most difficult to align because there is only one valid orientation for a successful insertion. In contrast, our method is successful on half-pipe 50% of the time. Upon further analysis, we notice that this is because GenCHiP generates a successful policy in only one direction of rotation (i.e., 100% successful for one rotation and 0% successful for the other randomized rotation).

When we inspect the code output from the LLM, we find that it generates intuitive waypoints for the search that correspond to the object specified in the prompt. For example, for the half-pipe, the output waypoints oscillate between 0 and $\frac{\pi}{2}$ while the star shape policy goes through multiples of $\frac{\pi}{4}$. We also find that the zero-shot prompting setup is sufficient for this application. Out of the box, the language model is able to parameterize the conditional compliant move.

	Cable Unroute	Cable Route	Connector Insertion	Connector Insertion (Perception)
CaP [1] (Few-Shot)	40%	0%	0%	0%
GenCHiP, FC (Few-Shot)	80%	30%	20%	0%
GenCHiP (Zero-Shot)	60%	0%	0%	0%
GenCHiP (Few-Shot)	**90%**	**100%**	**90%**	**60%**

TABLE II: **RGMC [13] and NTB [14] Assembly Tasks.** On challenging manufacturing tasks, it's critical to have compliance and conditional terminations in the action space. GenCHiP outperforms other LLM-based code-generation approaches for this reason.

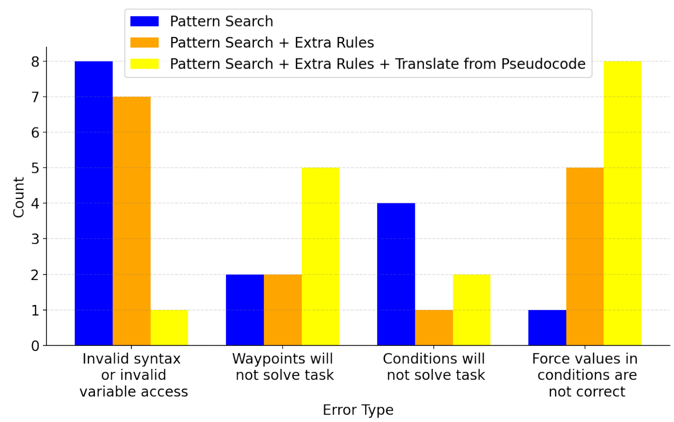


Fig. 7: We classify error types across three levels of hints. The severity of errors decreases from left to right. After combining all hints, the most common error is correct specification of force values.

Industrial Manipulation Tasks. To validate our method on more realistic force-based manipulation environments, we turn to the IROS RGMC [13] and NIST Assembly [14] experiments, which are arguably more directly targeted towards force-based manipulation than the peg insertion task. Indeed, we find it difficult to design an analogous scripted policy baseline for cable (un-)routing that would perform well across both tasks, which is why we omit it in this experiment. From the results in Table II, we observe that GenCHiP (Few-Shot) again consistently outperforms the baselines. Fixed Compliance (Few-Shot) is second-best, while Code-as-Policies [1] (Few-Shot) performs worst, failing to complete the routing task even a single time. The Zero-Shot version of our method also fails for these tasks, and for two reasons: (1) the LLM tends to generate `while` loops that are incompatible with the way the API is structured and (2) the program is successful within a narrow range of force values that are difficult to infer without more information from examples.

VI. Conclusions

In this work, we study the capability of Large Language Models (LLMs) to generate policies for a variety of high-precision, contact-rich manipulation tasks. By allowing LLMs to place constraints on robot impedances and interaction forces, GenCHiP improves success rates on subtasks derived from the Functional Manipulation Benchmark (FMB) and NIST Task Boards by 3x and 4x, respectively, when compared to code generation approaches that don't allow for compliance. This is the first work to show that robotic code generation using language models can yield policies that are successful in completing contact-rich manipulation tasks.

References

[1] J. Liang, W. Huang, F. Xia, *et al.*, "Code as policies: Language model programs for embodied control," in *arXiv preprint arXiv:2209.07753*, 2022.

[2] S. Mirchandani, F. Xia, P. Florence, *et al.*, "Large language models as general pattern machines," *arXiv preprint*, Jul. 2023.

[3] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *arXiv preprint arXiv:2201.07207*, 2022.

[4] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *arXiv preprint arXiv:2306.17582*, 2023.

[5] Y. Jiang, A. Gupta, Z. Zhang, *et al.*, "Vima: General robot manipulation with multimodal prompts," *arXiv preprint arXiv:2210.03094*, 2022.

[6] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on Robot Learning*, 2022.

[7] A. Brohan, N. Brown, J. Carbajal, *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[8] A. Brohan, N. Brown, J. Carbajal, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[9] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*, 2023.

[10] S. Belkhale, T. Ding, T. Xiao, *et al.*, *Rt-h: Action hierarchies using language*, 2024. arXiv: 2403.01823 [cs.RO].

[11] K. Tracy, Z. Manchester, A. Jain, *et al.*, "Efficient online learning of contact force models for connector insertion," *arXiv preprint arXiv:2312.09190*, 2023.

[12] J. Luo, C. Xu, L. Tan, L. Lin, J. Wu, and S. Levine, *Fmb: A functional manipulation benchmark for generalizable robotic learning*, 2023.

[13] Y. Sun, J. Falco, M. A. Roa, and B. Çalli, "Research challenges and progress in robotic grasping and manipulation competitions," *IEEE Robotics and Automation Letters*, vol. 7, pp. 874–881, 2021.

[14] K. Kimble, K. V. Wyk, J. Falco, *et al.*, "Benchmarking protocols for evaluating small parts robotic assembly systems," *IEEE Robotics and Automation Letters*, vol. 5, pp. 883–889, 2020.

[15] W. Huang, F. Xia, T. Xiao, *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *arXiv preprint arXiv:2207.05608*, 2022.

[16] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *Autonomous Robots*, 2023.

[17] I. Singh, V. Blukis, A. Mousavian, *et al.*, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[18] J. Wu, R. Antonova, A. Kan, *et al.*, "Tidybot: Personalized robot assistance with large language models," *Autonomous Robots*, 2023.

[19] A. Xie, Y. Lee, P. Abbeel, and S. James, "Language-conditioned path planning," in *Conference on Robot Learning*, PMLR, 2023, pp. 3384–3396.

[20] Y. Gai, P. Jain, W. Zhang, J. Gonzalez, D. X. Song, and I. Stoica, "Grounded graph decoding improves compositional generalization in question answering," *ArXiv*, vol. abs/2111.03642, 2021.

[21] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[22] Y. Wi, M. V. der Merwe, P. Florence, A. Zeng, and N. Fazeli, "CALAMARI: Contact-aware and language conditioned spatial action MApping for contact-RIch manipulation," in *7th Annual Conference on Robot Learning*, 2023.

[23] J. Luo, E. Solowjow, C. Wen, *et al.*, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019. DOI: 10.1109/ICRA.2019.8793506.

[24] T. Z. Zhao, J. Luo, O. Sushkov, *et al.*, "Offline meta-reinforcement learning for industrial insertion," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022.

[25] Y. Narang, K. Storey, I. Akinola, *et al.*, "Factory: Fast contact for robotic assembly," *arXiv preprint arXiv:2205.03532*, 2022.

[26] Z. Liu, Q. Liu, W. Xu, L. Wang, and Z. Zhou, "Robot learning towards smart robotic manufacturing: A review," *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102 360, 2022.

[27] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *J. Mach. Learn. Res.*, vol. 22, no. 1, Jan. 2021, ISSN: 1532-4435.

[28] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104 224, 2022.

[29] Í. Elguea-Aguinaco, A. Serrano-Muñoz, D. Chrysostomou, I. Inziarte-Hidalgo, S. Bøgh, and N. Arana-Arexolaleiba, "A review on reinforcement learning for contact-rich robotic manipulation tasks," *Robotics and Computer-Integrated Manufacturing*, 2023.

[30] A. S. Morgan, B. Wen, J. Liang, A. Boularias, A. M. Dollar, and K. Bekris, "Vision-driven compliant manipulation for reliable, high-precision assembly tasks," *arXiv preprint arXiv:2106.14070*, 2021.

[31] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy, "Residual learning from demonstration: Adapting dmps for contact-rich manipulation," *IEEE Robotics and Automation Letters*, 2022.

[32] T. Migimatsu, W. Lian, J. Bohg, and S. Schaal, "Symbolic state estimation with predicates for contact-rich manipulation tasks," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022.
[33] C. Chi, S. Feng, Y. Du, *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.
[34] G. Schoettler, A. Nair, J. Luo, *et al.*, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5548–5555, 2019.
[35] S. Brahmbhatt, A. Deka, A. Spielberg, and M. Müller, "Zero-shot transfer of haptics-based object insertion policies," in *International Conference on Robotics and Automation (ICRA)*, Jun. 2023.
[36] C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, and K. Harada, "Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach," *Applied Sciences*, vol. 10, no. 19, p. 6923, 2020.
[37] F. J. Abu-Dakka and M. Saveriano, "Variable impedance control and learning—a review," *Frontiers in Robotics and AI*, vol. 7, p. 590 681, 2020.
[38] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *ArXiv*, vol. abs/2205.11916, 2022.
[39] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
[40] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, 2011.
[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.
[42] W. Yu, N. Gileadi, C. Fu, *et al.*, "Language to rewards for robotic skill synthesis," *Arxiv preprint arXiv:2306.08647*, 2023.
[43] OpenAI, "Gpt-4 technical report," 2023.