DiffVL: Scaling Up Soft Body Manipulation using Vision-Language Driven Differentiable Physics

Zhiao Huang[*]
Computer Science & Engineering
University of California, San Diego
z2huang@ucsd.edu

Feng Chen[*]
Institute for Interdisciplinary Information Sciences
Tsinghua University
chenf20@mails.tsinghua.edu.cn

Yewen Pu
Autodesk
yewen.pu@autodesk.com

Chunru Lin
UMass Amherst
chunrulin@umass.edu

Hao Su[†]
University of California, San Diego
haosu@ucsd.edu

Chuang Gan[†]
MIT-IBM Watson AI Lab, UMass Amherst
chuangg@umass.edu

Abstract

Combining gradient-based trajectory optimization with differentiable physics simulation is an accurate and efficient technique for solving soft-body manipulation problems. Using a well-crafted optimization objective, the solver can quickly converge onto a valid trajectory. However, writing the appropriate objective functions requires expert knowledge, making it difficult to collect a large set of naturalistic problems from non-expert users. We introduce DiffVL, a framework that integrates the process from task collection to trajectory generation leveraging a combination of visual and linguistic task descriptions. A DiffVL task represents a long horizon soft-body manipulation problem as a sequence of 3D scenes (key frames) and natural language instructions connecting adjacent key frames. We built GUI tools and tasked non-expert users to transcribe 100 soft-body manipulation tasks inspired by real-life scenarios from online videos. We also developed a novel method that leverages large language models to translate task language descriptions into machine-interpretable optimization objectives, which can then help differentiable physics solvers to solve these long-horizon multistage tasks that are challenging for previous baselines. Experiments show that existing baselines cannot complete complex tasks, while our method can solve them well. Videos can be found on the website https://sites.google.com/view/diffvl/home.

1 Introduction

This paper focuses on soft body manipulation, a research topic with a wide set of applications such as folding cloth [52, 90, 34], untangling cables [88, 87], and cooking foods [77, 72]. Due to their complicated physics and high degree of freedom, soft body manipulations raise unique opportunities and challenges. Recent works such as [32] and [99] have heavily leveraged various differentiable physics simulators to make these tasks tractable. Towards generalizable manipulation skill learning, such approaches have the potential to generate data for learning from demonstration

---

[*]Equal contribution
[†]Equal contribution

algorithms [51, 43]. However, to enable differentiable physics solvers to generate a large scale of meaningful trajectories, we must provide them with suitable tasks containing the scene and the optimization objectives to guide the solver. Previous tasks are mostly hand-designed [32] or procedure-generated [51], resulting in a lack of diverse and realistic soft-body manipulation tasks for researchers.

This paper takes another perspective by viewing tasks as data, or more precisely, task specifications as data. Each data point contains a pair of an initial scene and an optimization objective depicting the goal of the task. Taking this perspective, we can annotate meaningful tasks like annotating data of other modalities, like texts, images, videos, or each action trajectory, by leveraging non-expert human labor, providing possibilities for scaling up the task space of differentiable physics solvers.

A core problem in building a task collection framework with trajectory generation is finding a suitable representation of the tasks. The representation should be intuitive for non-expert annotators while being accurate enough to describe the complex shapes and motions of the soft bodies that appear in various soft body manipulation processes. Last but not least, it should be reliably interpreted by differentiable physics solvers to yield a valid trajectory.
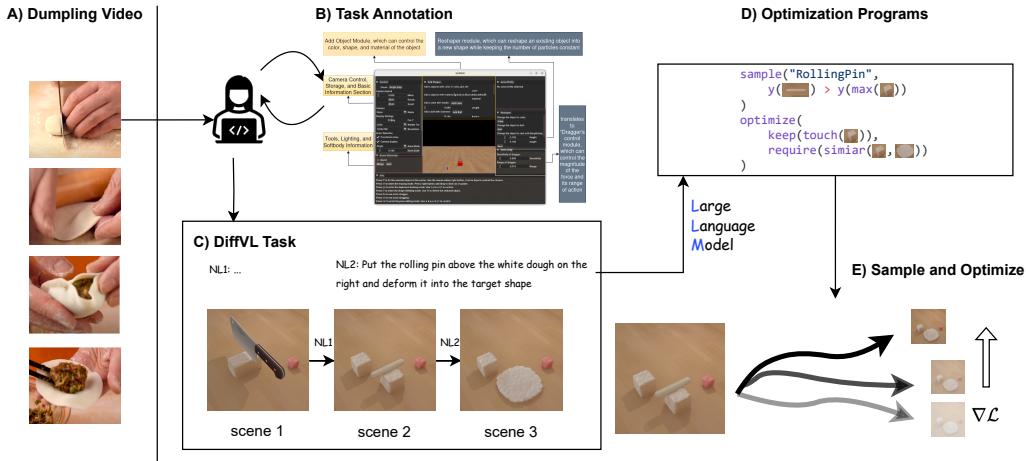

Figure 1: (A) A dumpling making video; (B) The annotator interacts with our GUI tool to create DiffVL tasks; (C) A DiffVL task contains a sequence of 3D scenes along with natural language instructions to guide the solver; (D) DiffVL leverages a large language model to compile instructions into optimization programs consisting of vision elements; (E) The optimization program guides the solver to solve the task in the end.

Studies such as [3, 1, 60, 89] have shown that humans are proficient at defining goal-driven, spatial object manipulation tasks using either sketches, natural languages, or both. Figure 1(A), for instance, depicts the complete process of dumpling creation, supplemented by textual instructions. Taking this as an inspiration, we present DiffVL, a framework that enables non-expert users to specify soft-body manipulation tasks to a differentiable solver using a combination of vision and language. Specifically, each DiffVL task consists of a sequence of 3D scenes (keyframes), with natural language instruction connecting adjacent keyframes. The sequence of key frames specifies the sequence of *subgoals* of the manipulation task, and the natural language *instructions* provides suggestions on how to use the actuators to manipulate the objects through this sequence of subgoals. See Figure 1.

We develop tools to ease the annotation for non-expert users. With our interactive simulator equipped with a GUI, the user can edit, manipulate, draw, or carve shapes easily like other 3D editing tools and observe the consequence through simulation in an intuitive manner. Meanwhile, when it is tedious for users to edit all the intermediate steps of a complex motion, they can use natural language to describe the goal instead of drawing them step by step. This enables us to build **SoftVL100**, a vision-language dataset with 100 diverse tasks. DiffVL uses LLM to compile the natural language instructions to an optimization program – which enforces a set of constraints during the soft-body manipulations from one keyframe to the next. This optimization program is then used by a differentiable physics solver to generate a working trajectory.

To summarize, our work makes the following contributions:

- We propose a new multi-stage vision-language representation for defining soft-body manipulation tasks that is suitable for non-expert user annotations.
- We developed a corresponding GUI, and curated SoftVL100, consisting of 100 realistic soft-body manipulation tasks from online videos[3].
- We develop a method, DiffVL, which marries the power of a large-language model and differentiable physics to solve a large variety of challenging long-horizon tasks in SoftVL100.

2 Related Work

Differentiable simulation for soft bodies Differentiable physics [30, 93, 69, 19, 29] has brought unique opportunities in manipulating deformable objects of various types [58, 28, 53, 33, 92, 49, 95, 83, 63, 100, 95, 56, 27, 47, 74, 23], including plasticine [32], cloth [45], ropes [54] and fluids [96] simulated with mass-spring models [30], Position-based Dynamics [65], Projective Dynamics [69, 16], MPM [36] or Finite Element Method [26], which could be differentiable through various techniques [30, 16, 57, 7]. It has been shown that soft bodies have smoother gradients compared to rigid bodies [32, 30, 93, 6, 82], enabling better optimization. However, the solver may still suffer from non-convexity [44] and discontinuities [82], motivating approaches to combine either stochastic sampling [44], demonstrations [43, 6] or reinforcement learning [98, 64] to overcome the aforementioned issues. DiffVL capitalizes on an off-the-shelf differentiable physics solver to annotate tasks. It incorporates human priors through a novel vision-language task representation. This allows users to employ natural language and GUI to guide the differentiable solver to solve long-horizon tasks, distinguishing DiffVL from previous methods.

Task representation for soft body manipulation There are various ways of defining tasks for soft body manipulation. To capture the complex shape variations of soft bodies, it is natural to use either RGB images [95, 51, 73] or 3D point cloud [78, 47] to represent goals. However, given only the final goal images, it may be hard for the solver to find a good solution without further guidance. Besides, generating diverse images or point clouds that adhere to physical constraints like gravity and collision avoidance in varied scenes can be challenging without appropriate annotation tools. Our GUI tools are designed to address these issues, simplifying the process. Other works use hand-defined features [10, 88] or formalized languages [41]. These methods necessitate specialist knowledge and require tasks to be defined on a case-by-case basis. Learning from demonstrations [43, 73] avoids defining the task explicitly but requires agents to follow demonstration trajectories and learn to condition on various goal representations through supervised learning or offline reinforcement learning [20, 18, 79, 24, 55, 35, 11, 2, 5]. However, collecting demonstrations step-by-step [70, 101, 43] requires non-trivial human efforts, is limited to robots with human-like morphologies, and may be challenging for collecting tasks that are non-trivial for humans (e.g., involving complex dynamics or tool manipulation). Our annotation tool emphasizes the description of tasks over the identification of solutions. Annotators are merely required to define the task, not execute the trajectories themselves, thereby simplifying the annotation process. Our method is also related to [46], but we consider a broader type of tasks and manipulation skills.

Language-driven robot learning Many works treat language as subgoals and learn language-conditioned policies from data [80, 21, 66, 42, 8, 62, 61] or reinforcement learning [37, 17, 84, 12]. It has been shown that conditioning policies on languages lead to improved generalization and transferability [80, 37]. [76] turns languages into constraints to correct trajectories. Our method distinguishes itself by integrating language with differentiable physics through machine-interpretable optimization programs. Recent works aim at leveraging large language models [4, 15, 94, 50, 13] to facilitate robot learning. Most use language or code [48] to represent policies rather than goals and focus on high-level planning while relying on pre-defined primitive actions or pre-trained policies for low-level control. In contrast, our language focuses on low-level physics by compiling language instructions to optimization objectives rather than immediate commands. It's noteworthy that VIMA [38] also introduces multimodal prompts, embodying a similar ethos to our vision-language task representation. Their approach views multimodal representation as the task prompts to guide the policy. However, it still requires pre-programmed oracles to generate offline datasets for learning, which is hard to generalize to soft-body tasks with complex dynamics. Contrastingly, we

---

[3]both the GUI and dataset will be made public

adopt a data annotation perspective, providing a comprehensive framework for task collection, natural language compiling, and trajectory optimization, enabling us to harness the potential of differentiable physics effectively.

Task and motion planning Our optimization program compiled from natural language is closely connected to the field of Task and Motion Planning (TAMP) [22], particularly Logic Geometric Programming (LGP) [85, 86]. We draw inspiration from LGP's multiple logical states in designing our DiffVL task representation. However, unlike LGP, our approach asks human annotators to assign "logic states" and facilitates the manipulation of soft bodies through our novel vision-language representation. Some recent works have explored the intersection of TAMP and language models [14, 81, 59] while primarily focusing on generating high-level plans.

3 SoftVL100: Vision-Language Driven Soft-body Manipulation Dataset

Put the rolling pin above the white and manipulate the white into goal.
Then grasp the yellow by gripper and place them into goal places.
Finally grasp the white and wrap it as well as the yellow in it.

Grasp the back end of rope. Then Wind the rope and place the salmon ball.
Grasp the front end of rope. Then Wind the rope and place the black ball.

Grasp the front end of top yellow by gripper and place it on the bottom.
Then grasp the ropes by gripper to wind and place them into theirs goal shape.
Finally, grasp the red by gripper to place it on the top.
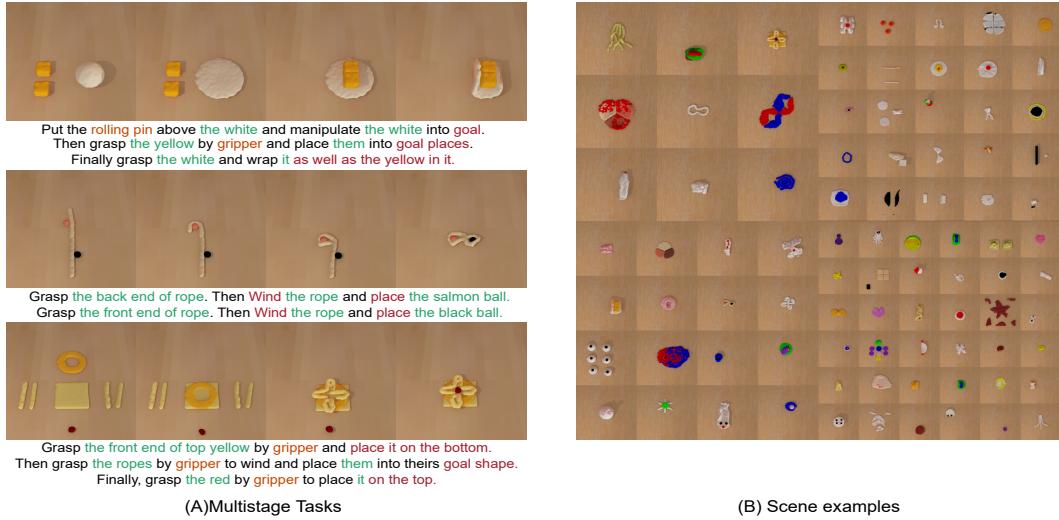
(A)Multistage Tasks

(B) Scene examples

Figure 2: (A) Example of multi-stage tasks and their text annotations; (B) Snapshots of scenes in SoftVL100 dataset.

In this representation, the original tasks are divided into a sequence of 3D scenes, or key frames, along with text descriptions that detail the steps for progressing to the next scene. Figure 2(A) illustrates our representation. Each key frame consists of multiple soft bodies with different positions and shapes. By grouping two consecutive key frames together, we form a manipulation stage. Natural language instructions may guide the selection of an actuator and provide guidance on how to control it to reach the subsequent key frames. In Section 3.1, we develop annotation tools to facilitate the task representation annotation process. These tools make it easy to construct a diverse set of tasks, referred to as SoftVL100, as discussed in Section 3.2.

3.1 Vision-Language Task Annotator

Our annotation tool is based on PlasticineLab [32], a simulation platform that utilizes the Material Point Method (MPM) [31] for elastoplastic material simulation. To enhance the user experience and support scene creation, we integrate it into SAPIEN [97], a framework that enables the development of customized graphical user interfaces.

Our simulation runs on GPU servers and the interface is accessible as a web service through VNC, allowing users to interact with the simulator directly from their web browser. When

Select the object to edit.   Add a new sphere   Cut it into two objects   Press it into a cylinder   Lift it up

The chosen sub-goal

Figure 3: Example operations in GUI tools

4

starting a new task annotation, users can open the simulator, perform actions, and save the current scene. Our GUI tools provide comprehensive support for creating, editing, managing, and simulating objects, as illustrated in Figure 3. **Scene creation:** The GUI offers a variety of primitive shapes, such as ropes, spheres, cubes, and cylinders, that users can add to the scene. Before adding a soft-body shape, users can adjust its size, color, and materials (e.g., rubber, fiber, dough, iron) if needed. **Shape editing:** Users have the ability to select, edit, or delete shapes within the scene. The GUI tool includes a range of shape operators, including moving, rotating, carving, and drawing. We provide a complete list of supported operations in the appendix. **Simulation:** Since the interface is built on a soft body simulator, users can simulate and interact with the soft body using rigid actuators or magic forces. They can also run simulations to check for stability and collision-free states. **Object management:** Users can merge connected shapes to create a single object or divide an object into two separate ones with distinct names. The annotation tool keeps tracking the identities of objects across scenes.

Once users have finished editing the scene and creating a new key frame, they can save the scene into the key frame sequence. The key frames are displayed below the simulator, visualizing the current task annotation progress. Users have the option to open any of these saved scenes in the simulator, delete a specific scene if needed, and add text annotations for each scene. Please refer to the appendix for detailed information on the functionalities and usage of the GUI tool.

3.2   The SoftVL100 Dataset

Our annotation tool simplifies the task creation process, enabling us to hire non-expert users to collect new tasks to form a new task set, SoftVL100. The task collection involves several stages, starting with crawling relevant videos from websites like YouTube that showcase soft body manipulation, particularly clay-making and dough manipulation. We developed tools to segment these videos and extract key frames to aid in task creation.

We hired students to annotate tasks, providing them with a comprehensive tutorial and step-by-step guidelines for using our annotation tools. It took approximately two hours for each annotator to become proficient in using the annotation tool. Subsequently, they were assigned a set of real-world videos to create similar tasks within the simulator. After collecting the keyframes, we proceed to relabel the scenes with text descriptions. Annotators are provided with a set of example text descriptions. We encourage users to include detailed descriptions of the actuators, their locations, moving directions, and any specific requirements, such as avoiding shape breakage for fragile materials. The annotation process for each task typically takes around 30 minutes. Using our task annotation tool, we have created a dataset called SoftVL100, which consists of 100 tasks, and there are more than 4 stages on average. The tasks cover a wide range of skills as illustrated in Figure 2 (A). Sample tasks from the dataset can be seen in Figure 2 (B).

4   Optimization with Vision-Language Task Description

We propose DiffVL to tackle the challenging tasks in SoftVL100. Given a stage of a task depicted in Figure 4(A), DiffVL utilizes large language models to compile the natural language instructions into a machine-interpretable optimization program in Figure 4(B). The program comprises of the names of visible elements and Python functions, effectively capturing the essence of the language instructions. We introduce the design of our DSL in Section 4.1 and outline the DiffVL compiler based on large language models (LLM) in Section 4.2. The resulting optimization program includes crucial information for selecting and locating actuators and can facilitate a differentiable physics solver to generate valid trajectories, as discussed in Section 4.3.

4.1   Optimization Program

The optimization program is formulated using a specialized domain-specific language (DSL) that incorporates several notable features. Firstly, it includes functions that extract the names of visible elements from the 3D scenes and support operations on these elements, represented as 3D point clouds. This enables manipulation and analysis of the visual information within the optimization program. Secondly, the DSL provides various functions to express geometric and temporal relations, encompassing common geometric and motion constraints. These functions facilitate the representation and handling of natural language instructions. Furthermore, the DSL clauses are interpreted
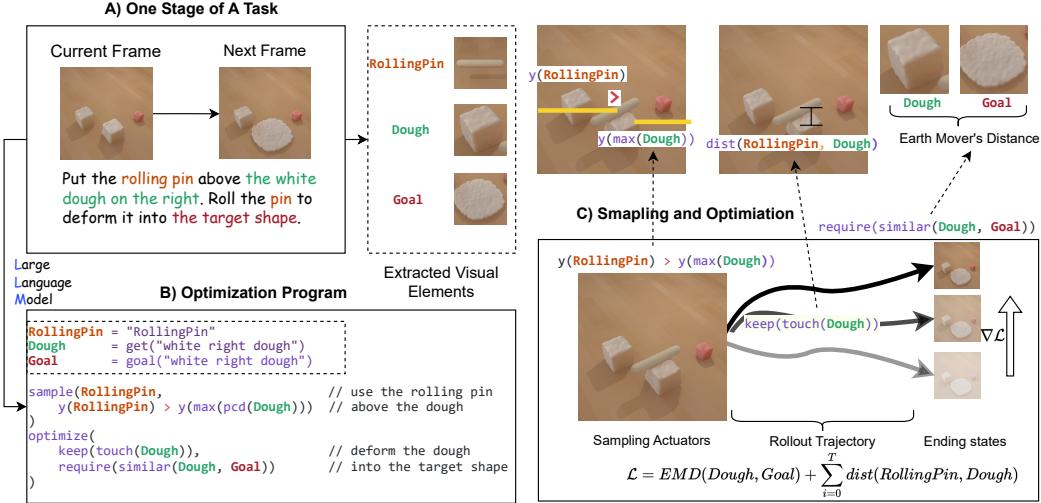
Figure 4: (A) One stage of the DiffVL task of two keyframes and a natural language instruction. (B) The compiled optimization program. (C) The sampling and optimization process.

into PyTorch [68], making the constraints automatically differentiable and directly applicable for differentiable physics solvers.

The example in Figure 4 already exposes multiple components of our DSL. In the program, we can refer to the names of visible elements through `get` and `goal` functions, which can look for and extract the 3D objects that satisfy the description, identified by their color, shapes, and locations (we ensured that these attributes are sufficient to identify objects in our tasks). The program will then start with a `sample` statement, which is used to specify the actuators and their associated constraints over initial positions. Actuators refer to the entities utilized for manipulation, such as a robot gripper, a knife, or the rolling pin depicted in Figure 4. The constraint `y(RollingPin) > y(max(pcd(Dough)))` within the `sample` statement represents a specific constraint. It ensures that the height of the actuator (RollingPin) is greater than the highest y coordinate of the white dough in the scene, reflecting the meaning of "put bove" of the language instructions. Following the `sample` statement, an `optimize` statement is included. This statement plays a vital role in defining the constraints and objectives for the manipulation trajectories. These constraints and objectives serve as the differentiable optimization objectives that the differentiable physics solver aims to maximize. In an optimization program, two special functions, namely `require` and `keep`, can be used to specify the temporal relationship of the optimization objectives. By default, the `require` function evaluates the specified condition at the end of the trajectories. On the other hand, the `keep` function is applied to each frame of the trajectories, ensuring that the specified conditions are maintained throughout the trajectory. In the example program shown in Figure 4(B), the optimization program instructs the actuator to continuously make contact with the white dough. The function `similar` computes the Earth Mover distance [71] between two objects' point clouds, which aims to shape the dough into a thin pie, the goal shape extracted from the next key frame. We illustrate more example optimization programs in Figure 5.

4.2 Compiling Natural Languages with LLM

We utilize the power of the Large Language Model (LLM) to compile text instructions into optimization programs. This approach capitalizes on the few-shot capabilities of the LLM and employs various few-shot prompts [91, 9]. The prompts begin by defining the "types" and "functions" within our DSL. Each function is accompanied by a language explanation to clarify its purpose. Additionally, examples are given to demonstrate how primitive functions can be combined to form complex clauses. To facilitate the translation, we provide the language model with both the scene description of the

A) Grasp the front of the rope and then grasp the back of the rope. Deform the rope into its target shape.

```
sample("gripper", grasp(frontend(get("rope"))))
optimize(require(emd(get("rope"), goal("rope"))))
sample("gripper", grasp(backend(get("rope"))))
optimize(require(similar(get("rope"), goal("rope"))))
```

B) Use the gripper to first lift up the white cube above the black wall and then move it to the target location

```
sample("gripper", grasp(get("white")))
optimize(
  require(y(com("white"))>y(max(pcd("black"))), 0.5),
  require(l2(com(get("white")), com(goal("white")))),
)
```

C) Put the knife above the white object. Cut the knife down to match the target shape of the white object. Ensure that the knife does not rotate.

```
sample("knife", roll(0.),
       y("knife") > y(max(pcd("white"))))
optimize(
  keep(touch(leftpart("white"))),
  keep(roll() < 0.1), keep(pitch() < 0.1),
  require(similar("white", goal("white"))),
)
```

D) Pick up the front part of object 'top left mat' and place it into the goal without breaking it.

```
sample("gripper", touch(frontpart("top left mat"))
optimize(
  keep(touch(frontpart("top left mat"))),
  require(similar("top left mat",
    goal("top left mat"))),
  keep(no_break("top left mat")),
)
```
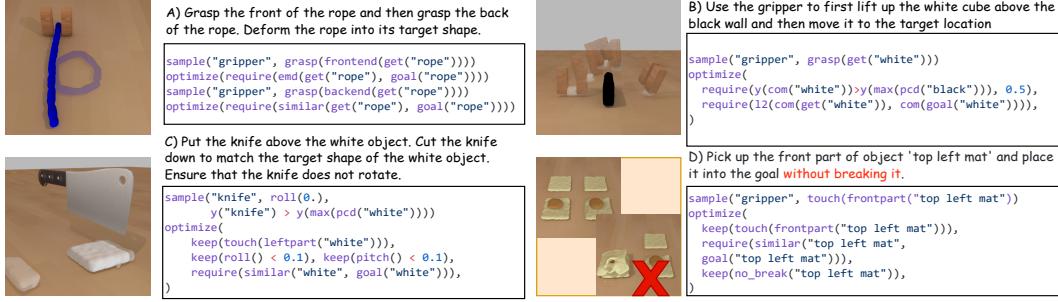
Figure 5: A) Decomposing a stage into sub-stages: manipulating a rope into a circle using a single gripper requires the agent to first grasp the upper part of the rope and then manipulate the other side to form the circle. The program counterpart contains two pairs of `sample` and `optimize`, reflecting the two sub-stages in the language instruction. `backend` extracts the back end of 3D objects; B) Guiding the motion of the objects to avoid local optima: we use `com` to compute the center of the mass of objects and `pcd` to get the clouds; relation operators compare two objects' coordinate. The first `require` takes an additional argument 0.5, meaning the inner clause, which asks the white object above the black one, should be evaluated halfway through the trajectory to represent meanings of 'first do A and then do B.' C) Selecting a suitable tool to split the objects; D) The program can include additional constraints like not breaking the shape, which is critical for manipulating fragile materials.

objects that it contains and the natural language instructions that we wish to compile. We then prompt the model with the instruction, "Please translate the instructions into a program." This step is repeated for all frames, resulting in their respective optimization programs. Through our experimentation, we have observed that GPT-4 [67] surpasses other models in terms of performance. Please refer to the appendix for a more comprehensive understanding of the translation process.

4.3 Solving the Optimization through Differentiable Physics

We leverage the differentiable physics solver in [32], which has been proven accurate and efficient in solving the generated optimization programs. The process is illustrated in Figure 4(C). The `sample` function is employed within a sampling-based motion planner. Initially, it samples the pose of the specified actuator type to fulfill the provided constraints. Subsequently, an RRT planner [40] determines a path toward the target location, and a PD controller ensures that the actuators adhere to the planned trajectory. With the actuator suitably initialized, the `optimize` clause is then passed to a gradient-based optimizer to optimize the action sequence for solving the task. The solver performs rollouts, as depicted in Figure 4(C), where the conditions enclosed by `keep` operators are evaluated at each time step. Conversely, conditions enclosed by `require` are evaluated only at specified time frames. The resulting differentiable values are accumulated to calculate a loss, which is utilized to compute gradients for optimizing the original trajectories. For handling multistage tasks involving vision-language representation, we can solve them incrementally, stage by stage.

5 Experiments

In this section, we aim to justify the effectiveness of our vision-language task representation in guiding the differentiable physics solver. For a detailed analysis of the ablation study on the language translator, we direct readers to the appendix. To better understand the performance of different solvers, we evaluate the tasks of two tracks. The first track focuses on tasks that only require agents to transition from one key frame to another. Tasks in this track are often short-horizon. This setup allows us to evaluate the efficiency and effectiveness of the differentiable physics solver in solving short-horizon soft body manipulation tasks, as described in Section 5.2. In the second track, we validate the agent's capability to compose multiple key frames in order to solve long-horizon manipulation tasks.

These tasks involve complex scenarios where agents need to switch between different actuators and manipulate different objects. The evaluation of this composition is discussed in Section 5.3.

5.1 Initialization for tasks

We clarify that our approach does not use a gradient-based optimizer to optimize trajectories generated by the RRT planner. Motion planning and optimization occur in separate temporal phases. Initially, we plan and execute a trajectory to position the actuator at the specified initial pose. Subsequently, we initialize a new trajectory starting from this pose, and the actuator remains fixed in this trajectory as we initialize the policy with zero actions to maintain its position. The sample function generates poses in line with the annotators' guidance.

5.2 Mastering Short Horizon Skills using Differentiable Physics

Before evaluating the full task set, we first study how our vision-language representation can create short-horizon tasks that are solvable by differentiable physics solvers. We picked 20 representative task stages as our test bed from the SoftVL100. We classify these tasks into 5 categories, with 4 tasks in each category. The categories are as follows: **Deformation** involves the deformation of a large single 3D shape, achieved through pinching and carving; **Move** asks the movement of an object or stacking one object onto another; **Winding** involves the manipulation of a rope-like object to achieve different configurations. **Fold** focuses on folding a thin mat to wrap other objects. **Cut** involves splitting an object into multiple pieces. The selected tasks encompass a wide spectrum of soft body manipulation skills, ranging from merging and splitting to shaping. We evaluate different methods' ability to manipulate the soft bodies toward their goal configuration. The agents are given the goal configuration in the next frame, and we measure the 3D IoU of the ending state as in [32]. We set a threshold for each scene's target IoU score to measure successful task completion, accommodating variations in IoU scores across different scenes.

We compare our method against previous baselines, including two representative reinforcement learning algorithms, Soft Actor-Critic (SAC) [25] and Proximal Policy Gradient (PPO) [75]. The RL agents perceive a colored 3D point cloud as the scene representation and undergo 10^6 steps to minimize the shape distance towards the goal configuration while contacting objects. Additionally, we compare our approach with CPDeform [44], which employs a differentiable physics solver. However, CPDeform uses the gradient of the Earth mover's distance as a heuristic for selecting the initial actuator position. It differs from DiffVL which utilizes the text description to determine ways to initialize the actuators and the optimization objectives. For differentiable physics solvers, we run Adam [39] optimization for 500 gradient steps using a learning rate of 0.02.

Table 1 presents the success rates and mean IoU of various approaches. It is evident that, except for specific deformation and cutting tasks where SAC shows success, the RL baselines struggle to solve most tasks. Although RL agents demonstrate an understanding of how to manipulate the soft bodies correctly, they lack the precision required to match the target shape accurately. Additional experimental results can be found in the appendix. CPDeform achieves partial success in each task category. It surpasses the RL baselines by utilizing a differentiable physics solver and employing a heuristic that helps identify contact points to minimize shape differences. However, CPDeform's heuristic lacks awareness of physics principles and knowledge about yielding objects to navigate around obstacles. Consequently, CPDeform fails to perform effectively in other tasks due to these limitations. We want to emphasize that integrating language into RL baselines entails non-trivial challenges. For example, many reward functions in our optimization program have temporal aspects, making the original state non-Markovian. As we focus on enhancing the optimization baseline through language-based experiments, we've chosen to defer tackling the complexities of language-conditioned RL to future research.

Our approach surpasses all other methods, achieving the highest overall success rate. To examine the impact of different components and illustrate their effects, we conduct ablation studies. First, we remove the constraints in `sample` named "- Sample" in the table and ask the agent to sample random tools without using hints from the language instructions. As a result, the performance drops significantly, either getting stuck at the local optima or failing to select suitable tools to finish the task. For example, in the cutting task, it failed to select knives and split the white material into two pieces. On the other hand, the variant "- Optimize" replaces the statement in `optimize` by

| | previous baselines | | | DiffVL | | |
| | SAC | PPO | CPDeform | - Sample | - Optimize | Ours |
	SR/IOU	SR/IOU	SR/IOU	SR/IOU	SR/IOU	SR/IOU
Deform	0.25 (0.504)	0.00/0.392	0.25/0.532	0.11/0.489	0.44/0.524	**1.00/0.564**
Move	0.00/0.541	0.00 /0.527	0.08 /0.570	0.00/0.533	0.33/0.618	**1.00/0.641**
Wind	0.00/0.308	0.00/0.289	0.21/0.362	0.25 /0.351	0.08 /0.408	**0.59/0.446**
Fold	0.00/0.572	0.00 /0.457	0.33/0.612	0.00/0.550	**1.00 /0.650**	0.94/0.643
Cut	0.33/0.449	0.00/0.408	**0.89/0.482**	0.33/0.357	0.56/0.447	0.87/0.490
Total	0.12/0.475	0.00/0.415	0.35/0.512	0.14/0.456	0.48/0.529	**0.88/0.557**

Table 1: Single stage experiment results. The metrics we use are Success Rate (SR) and 3D Intersection Over Union (IOU).
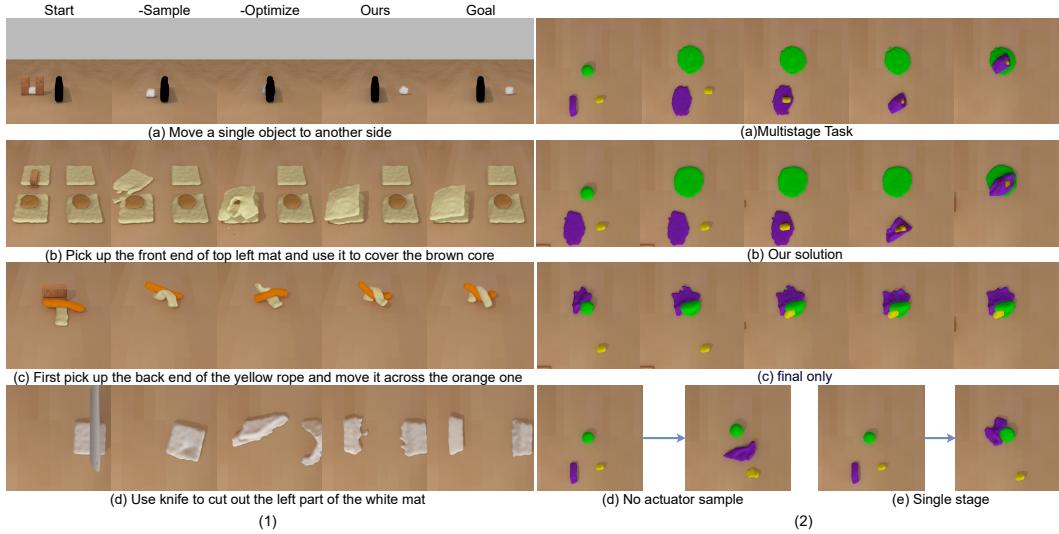
Figure 6: (1) Performance on single stage tasks. (2)Performance on a multistage task that includes moving, packing, and pressing.

an objective focusing solely on optimizing shape distance, removing additional guidance for the physical process. The effects are lesser compared to the previous variant, demonstrating that with a suitable tool initialization method, a differentiable physics solver is already able to solve many tasks. However, as illustrated in Figure 6(A), it does not lift the white cube to avoid the black wall in a moving task and breaks the mat shown in the second row. In the winding task, it fails to find the relatively complex lifting-then-winding plan directly from shape minimization. In contrast, our method can successfully solve these tasks after leveraging the trajectory guidance compiled from the natural language instructions. We find that our solver is quite efficient. For a single-stage task, it takes 10 minutes for 300 gradient descent steps on a machine with NVIDIA GeForce RTX 2080, for optimizing a trajectory with 80 steps. For most tasks, 300 gradient steps are sufficient for finding good solutions.

5.3 Driving Multi-Stage Solver using Vision-Language

Having evaluated short-horizon tasks, we apply our method to long-horizon tasks in SoftVL100. We leverage the natural language instructions to generate the evaluation metrics for each task, which might measure the relationships between objects, check if soft materials rupture, or if two shapes are similar. We refer readers to the appendix for more details.

Our multistage vision-language task representation decomposes the long-horizon manipulation process into multiple short-horizon stages, allowing our method to solve them in a stagewise manner. As baseline algorithms fail to make notable progress, we apply ablation to our method: We first compare

9

our method with a single-stage approach (single), which refers to solving long-horizon tasks like short-horizon tasks, by dropping the key frames and directly optimizing for the final goal. We then dropped the actuator sampling process in the middle (no actuator sample). The solver is still told to optimize for key frames (sub-goals) at each stage but was not informed to switch actuators and objects to manipulate. As expected, dropping the stage information and the language instructions in our vision task representation significantly degrades the performance, making solvers unable to complete most tasks.

After leveraging the language instructions, the FinalOnly solver removes the vision subgoals in each sub-stage but provides the agent's actuators to choose the objects to manipulate. In this case, similar to the "- Sample" variant, the differentiable physics solver can solve certain tasks by only minimizing shape distance. However, it may have troubles for tasks that require objects to deform into multiple shapes. For example, as illustrated in Figure 6(B), the solver needs to first compress the purple shape into a thin mat to warp the yellow dough.

Figure 7: Success rate of multi-stage tasks of different ablations

The solver fails to discover the compression process after removing the subgoals but directly moves the purple toward its final configurations, resulting in the failure in the end. This showcases the importance of the introduction of the key frames in our vision-language task annotation.
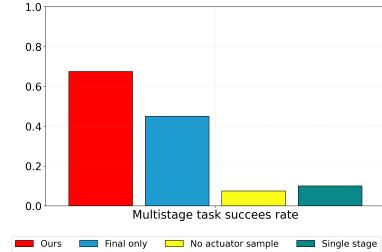
6 Limitations and Conclusion

We introduce DiffVL, a method that enables non-expert users to design specifications of soft-body manipulation tasks through a combination of vision and natural language. By leveraging large language models, task descriptions are translated into machine-interpretable optimization objectives, allowing differentiable physics solvers to efficiently solve complex and long-horizon multi-stage tasks. The availability of user-friendly GUI tools enables a collection of 100 tasks inspired by real-life manipulations. However, it is important to acknowledge some limitations. Firstly, the approach relies on human labor for dataset creation. Additionally, the utilization of large language models incurs high computational costs. Nevertheless, We anticipate being able to gather a larger set of tasks in the future. Furthermore, leveraging the generated trajectories to guide learning methods holds significant potential as a promising avenue for future exploration.

7 Future Works

We believe that crowdsourcing is vital in achieving this goal. We have taken the first step to demonstrate how non-experts can contribute to scaling up robot learning, which may inspire the field to explore ways of involving individuals from various backgrounds in expanding robot manipulation capabilities. It would be out of the capabilities of our group but can be readily adopted by other groups with better resources eventually.

Transferring the policies found by the differentiable simulator to the real world is important. We believe that various components of our framework can offer valuable contributions to real-world scenarios: one such contribution lies in our vision language representation, which naturally represents tasks in the real world; leveraging large language models aids in generating rewards for evaluating real-world results; our GUI tool can effectively manipulate real-world materials, particularly in constrained tasks, serving as a direct interface between humans and robot controllers.

8 Acknowledgements

References

[1] Samuel Acquaviva, Yewen Pu, Marta Kryven, Catherine Wong, Gabrielle E Ecanow, Maxwell Nye, Theodoros Sechopoulos, Michael Henry Tessler, and Joshua B Tenenbaum. Communicating natural programs to humans and machines. *arXiv preprint arXiv:2106.07824*, 2021.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[3] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (TOG)*, 22(3):828–837, 2003.

[4] 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.

[5] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[6] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. *arXiv preprint arXiv:2207.00167*, 2022.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[8] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[10] Lawrence Yunliang Chen, Baiyu Shi, Daniel Seita, Richard Cheng, Thomas Kollar, David Held, and Ken Goldberg. Autobag: Learning to open plastic bags and insert objects. *arXiv preprint arXiv:2210.17217*, 2022.

[11] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15084–15097. Curran Associates, Inc., 2021.

[12] Cédric Colas, Ahmed Akakzia, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Language-conditioned goal generation: a new approach to language grounding for rl. *arXiv preprint arXiv:2006.07043*, 2020.

[13] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for Dynamics and Control Conference*, pages 893–905. PMLR, 2022.

[14] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[15] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

[16] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021.

[17] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

[18] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *arXiv preprint arXiv:1910.13395*, 2019.

[19] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[20] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.

[21] Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. Lisa: Learning interpretable skill abstractions from language. *arXiv preprint arXiv:2203.00054*, 2022.

[22] 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.

[23] Narasimhan Gautham, Zhang Kai, Eisner Ben, Lin Xingyu, and Held David. Transparent liquid segmentation for robotic pouring. In *International Conference on Robotics and Automation (ICRA)*, 2022.

[24] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[25] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[26] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, July 2021.

[27] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *International Conference on Learning Representations*, 2020.

[28] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. VisuoSpatial Foresight for Physical Sequential Fabric Manipulation. *Autonomous Robots (AURO) journal*, 2021.

[29] Taylor A Howell, Simon Le Cleac'h, J Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 2022.

[30] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[31] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[32] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2020.

[33] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics model with occlusion reasoning for cloth manipulation. *Robotics: Science and Systems (RSS)*, 2022.

[34] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics with occlusion reasoning for cloth manipulation, 2022.

[35] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[36] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pages 1–52. 2016.

[37] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv: Arxiv-2210.03094*, 2022.

[39] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[40] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[41] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[42] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 35:31199–31212, 2022.

[43] Sizhe Li, Zhiao Huang, Tao Chen, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Dexdeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics. *arXiv preprint arXiv:2304.03223*, 2023.

[44] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Contact points discovery for soft-body manipulations with differentiable physics. *arXiv preprint arXiv:2205.02835*, 2022.

[45] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Transactions on Graphics (TOG)*, 42(1):1–20, 2022.

[46] Yulong Li, Shubham Agrawal, Jen-Shuo Liu, Steven K Feiner, and Shuran Song. Scene editing as teleoperation: A case study in 6dof kit assembly. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4773–4780. IEEE, 2022.

[47] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[48] 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.

[49] Junbang Liang, Ming C Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 2019.

[50] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[51] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B Tenenbaum, David Held, and Chuang Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *arXiv preprint arXiv:2203.17275*, 2022.

[52] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021.

[53] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021.

[54] Fei Liu, Entong Su, Jingpei Lu, Mingen Li, and Michael C Yip. Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics. *IEEE Robotics and Automation Letters*, 2023.

[55] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[56] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Trans. Graph.*, 37(4), July 2018.

[57] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[58] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010.

[59] Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. Pdsketch: Integrated domain programming, learning, and planning. *Advances in Neural Information Processing Systems*, 35:36972–36984, 2022.

[60] William P McCarthy, Robert D Hawkins, Haoliang Wang, Cameron Holdaway, and Judith E Fan. Learning to communicate about shared procedural abstractions. *arXiv preprint arXiv:2107.00077*, 2021.

[61] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.

[62] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

[63] Peter Mitrano, Dale McConachie, and Dmitry Berenson. Learning where to trust unreliable models in an unstructured world for deformable object manipulation. *Science Robotics*, 6(54):eabd8170, 2021.

[64] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021.

[65] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[66] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[67] OpenAI. Gpt-4 technical report, 2023.

[68] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[69] Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34:17123–17135, 2021.

[70] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *IEEE Robotics and Automation Letters*, 7(4):10873–10881, 2022.

[71] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99, 2000.

[72] Md. Sadman Sakib, David Paulius, and Yu Sun. Approximate task tree retrieval in a knowledge network for robotic cooking. *IEEE Robotics and Automation Letters*, 7(4):11492–11499, 2022.

[73] Gautam Salhotra, I-Chun Arthur Liu, Marcus Dominguez-Kuhne, and Gaurav S Sukhatme. Learning deformable object manipulation from expert demonstrations. *IEEE Robotics and Automation Letters*, 7(4):8775–8782, 2022.

[74] Connor Schenck and Dieter Fox. Visual closed-loop control for pouring liquids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2629–2636. IEEE, 2017.

[75] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[76] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *arXiv preprint arXiv:2204.05186*, 2022.

[77] Haochen Shi, Huazhe Xu, Samuel Clarke, Yunzhu Li, and Jiajun Wu. Robocook: Long-horizon elasto-plastic object manipulation with diverse tools, 2023.

[78] Haochen Shi, Huazhe Xu, Zhiao Huang, Yunzhu Li, and Jiajun Wu. Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks. *Robotics: Science and Systems (RSS)*, 2022.

[79] Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. *arXiv preprint arXiv:2207.07560*, 2022.

[80] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[81] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[82] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[83] Priya Sundaresan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E Gonzalez, and Ken Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9411–9418. IEEE, 2020.

[84] Allison Tam, Neil Rabinowitz, Andrew Lampinen, Nicholas A Roy, Stephanie Chan, DJ Strouse, Jane Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *Advances in Neural Information Processing Systems*, 35:25377–25389, 2022.

[85] 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.

[86] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. *Robotics: Science and Systems Foundation*, 2018.

[87] Vainavi Viswanath, Kaushik Shivakumar, Jainil Ajmera, Mallika Parulekar, Justin Kerr, Jeffrey Ichnowski, Richard Cheng, Thomas Kollar, and Ken Goldberg. Learning to trace and untangle semi-planar knots (tusk), 2023.

[88] Vainavi Viswanath, Kaushik Shivakumar, Justin Kerr, Brijen Thananjeyan, Ellen Novoseller, Jeffrey Ichnowski, Alejandro Escontrela, Michael Laskey, Joseph E Gonzalez, and Ken Goldberg. Autonomously untangling long cables. *arXiv preprint arXiv:2207.07813*, 2022.

[89] Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Chin-Yi Cheng, and Jiajun Wu. Translating a visual lego manual to a machine-executable plan. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVII*, pages 677–694. Springer, 2022.

[90] Yufei Wang, Zhanyi Sun, Zackory Erickson, and David Held. One policy to dress them all: Learning to dress people with diverse poses and garments, 2023.

[91] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[92] Thomas Weng, Sujay Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, 2021.

[93] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.

[94] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*, 2023.

[95] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations, 2020.

[96] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. *arXiv preprint arXiv:2303.02346*, 2023.

[97] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[98] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022.

[99] Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, Zhiao Huang, Chuang Gan, and Shuran Song. Roboninja: Learning an adaptive cutting policy for multi-material objects. *arXiv preprint arXiv:2302.11553*, 2023.

[100] Mengyuan Yan, Yilin Zhu, Ning Jin, and Jeannette Bohg. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, 5(2):2372–2379, 2020.

[101] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Supplementary files

We provide a video illustrating our GUI tools, example task, and solutions. We also provide a tutorial for helping users to use our annotation system as a supplementary file.

A   How to choose keyframes

The selection of keyframes is determined by the annotators, as we believe humans can naturally decompose complex tasks into simpler ones for communication. When annotators manipulate a scene within the GUI, they can save the current scene as a key frame within the task representation. They can also add, modify, or delete key frames using the editors within the web server interface.

The annotators were instructed on what kind of decomposition would likely result in a working trajectory, i.e., segmenting the manipulation processes at contact point changes., which is a simple and effective way for the annotator to provide high-quality labels, without having any expert understanding of the physics and the solver. If the agent is required to manipulate a new object or the actuator needs to establish contact on a different face of the object, annotators would introduce a new keyframe. Additionally, we employ heuristic methods to segment YouTube videos. This segmentation occurs when there is a significant disparity between two frames, aiming to simplify the keyframe selection for annotators.

B   DSL for Optimization Programs

Table 2 lists the functions and primitives in our domain-specific language.

We rely on the large language model's few-shot ability to compile the natural language description like "lift up the white cube above the black wall" into an executable optimization program. The prompt start with the introduction to the DSL, listing the functions as well as the explanations, followed by several examples illustrating how to compose those functions to implement complex functions, e.g., by checking the x coordinates of two objects to decide if one is on the left of the other. Then we provide several pairs of natural language inputs and corresponding programs. In the end, we provide the natural language input to compile. Below is an example prompt.

```
Here are the functions for obtaining objects and target objects. Their
    return types are all one single point-cloud object. A desc can
    only include strings that contain shape ["rope", "sphere", "box",
    "mat"], colors like ["white", "gray", "green", "red", "blue", "
    black"], and position like ["left", "right", "top", "bottom"]:
- get(desc) # Get the point cloud of the objects with the description,
     for example, get("left white mat"). If the desc is 'all', then
    get all the objects.
- goal(desc) # Get the point cloud of the goal for the objects with
    the given name. The name can also be 'all'.
- others(desc) # Get all other point clouds except the object with the
     given name.
- others_goals(desc) # Get all other point clouds of the goals except
    the object with the given name.

Here are the functions that use constraints. Note that the obj, and
    goal can only be one single point-cloud:
- keep(cond, start=s, end=t) # Minimize the reward from time s to time
     t. It only takes one cond as an argument.
- require(cond, end=t) # Reach the condition at time t. By default end
    =1.
- similar(obj, goal) # Compare the shape earth mover distance of the
    objects to the given goal shapes.
- touch(obj) # Minimize the distance between the tool and the object.
- fix_place(obj) # Ensure that the shape of the given object(s) and
    its positions do not change. It only takes one argument.
- no_break(obj) # Do not break the objects.
- away() # check if the actuator is away from all objects
```

Category	FuncName	Explanation
Objects	get(desc)	Get the point cloud of the objects with the description, for example, get("left white mat"). If the desc is 'all', then get all the objects.
	goal(descr)	Get the point cloud of the goal for the objects with the given name. The name can also be 'all'.
Temporal conditions	keep(cond, start=0, end=1)	Minimize the cond from time s to time t. It only takes one cond as an argument.
	require(cond, end=1)	Reach the condition at time t. By default end is 1.
	and(cond1, cond2, . . .)	Needs to satisfy all constraints
Shape operators	com(shape)	Compute the center of the mass of the point cloud
	similar(A, B)	Compare the shape distance using EMD distance
	pcd(shape)	Get the PyTorch tensor that represents the point cloud of the shape
	leftpart, rightend, etc.	Get a part of the point cloud based on the description.
Actuator	touch(ShapeA)	Minimize the signed distance function of the actuators and the shape
	away	Check if the actuator is far away from the shape
	roll, vertical, etc.	Get the rotation angle of the actuator
Tensor operators	Relation $>, <, \geq, \leq$	Compare relationships of two vectors or scalar
	Algebra $+, -$	Add values to something.
	l_2(A, B)	Compare the l_2 distance
	min, max	Get the min max of a tensor
Soft Body Constraints	fix_shape	Do not change the shape of the objects for too much
	fix_place	Keep the object not moving
	no_break	Do not break the objects
Special	stage(sample_fn, optimize_fn)	An optimization stage that may involve a motion planning procedure and an optimization procedure. It is only used for compiling.
	sample(tool_name, *conds)	Select the tool of tool_name. We provide "knife", "board", "rolling pin" and "gripper". The conds are list of conditions that we hope the tool satisfy
	optimize(*conds)	Optimize the trajectory to satisfy the conditions.

Table 2: Elements in the optimization program

```
- roll() # actuators ' rotation about x
- pitch() # actuator 's rotation about y
- yaw() # actuator 's rotation about y
- vertical() # actuator is vertical
- horizontal() # actuator is horizontal


Here are the example functions to obtain additional information:
- com(obj) # Center of the objects.
- pcd(obj) # point clouds of the objects.
- max(pcd(obj)) # max of the point cloud of the objects.
- min(pcd(obj)) # min of the point cloud of the objects.
- x(com(obj0)) # X coordinate of the points.
- x(min(pcd(obj0))) # smallest X coordinate of the boundary of the
    points.


We can compose those functions to implement various functions. Below
    are examples:
# The x coordinate of the objects
 x(com(obj0))

# Obj0 is on the left of obj1.
 lt(x(com(obj0)), x(com(obj1)))

# the front end of the Obj0 is above obj1.
 gt(y(com(frontend(obj0))), y(max(pcd(obj1))))

# Obj0 is in front of obj1.
 lt(z(com(obj0)), z(min(pcd(obj1))))

# deform the rope named by 'blue ' into its goal shape
 require(similar('blue ', goal('blue ')))

# deform object 'blue ' into its goal shape while fixing others
 require(similar('blue ', goal('blue '))), keep(fix_place(others('blue ')
    ))

# first do A then do B
 require(A, end=0.5), require(B, end=1.0)

# not rotate the actuator aboux x axis
 keep(roll() < 0.1)




Here are special function for
- stage(sample_clause, optimize_clause) # Each program may have
    multiple stage and each stage must have a 'sample ' function and an
     'optimize ' function. There are at most three stages. Start a new
    stage if and only if we need to change the actuators or manipulate
     different objects or parts.
- sample(actuator_name, *args) # sample the actuator with the given
    name in the beginning, args denotes for the conditions or
    requirements of the actuator 's pose.
- optimize(*args) # conditions needs to satiesfied during the
    manipulation.

Below are examples of the scenes and the program to solve the scenes.


Input: grasp the front end of the blue rope vertically and then deform
     into its goal pose and please do not break it. Then grasp the
```

```
     back of the mat and then pick up the mat and move it into its goal
      position.
Program:
blue_rope = get("blue rope")
stage(
    sample("gripper", grasp(frontend(blue rope)), vertical()),
    optimize(
        require(similar(blue_rope, goal('blue rope'))),
        keep(touch(blue_rope)),
        keep(no_break(blue_rope))
    )
)
mat = get("mat")
stage(
    sample("gripper", grasp(backpart(mat))),
    optimize(
        require(l2(com(get(mat)), com(goal("mat")))),
        keep(touch(backpart(mat)))
    )
)


Input: cut the mat into its goal shapes and then move the knife away.
Program:
mat = get("mat")
stage(
    sample("knife", touch(mat)),
    optimize(
        require(similar(mat, goal('mat'))),
        keep(touch(mat), end=0.5),
        require(away())
    )
)

Input: move the object 'A' to the left of 'B' then move it to the
    right.
Program:
A = get('A')
B = get('B')
stage(
    sample("gripper", grasp(A)),
    optimize(
        require(lt(x(com(A)), x(com(B))), end=0.5),
        require(gt(x(com(A)), x(com(B))), end=1),
        keep(touch(A))
    )
)


Input: Put the board above all objects and deform them into their goal
     shapes and keep them not broken
Program:
stage(
    sample("board", gt(y("board"), y(max(pcd('all'))))),
    optimize(
        require(similar('all', goal('all'))),
        keep(touch('all')),
        keep(no_break('all')),
    )
)


Input: Use the gripper to grasp the right part of the object on the
    right and deform it into its target shape while fixing others.
Program:
```

```
left = get('left')
stage(
    sample("gripper", rightpart(left)),
    optimize(
        require(similar(left, goal('left'))),
        keep(fix_place(others('left'))),
        keep(touch(rightpart(left))),
    )
)


Input: There are two objects of different colors "red", "green". Use
    the board to touch them one by one. Deform them into their target
    shape.
Program:
stage(
    sample("board", touch("red")),
    optimize(
        keep(touch('red')),
        require(similar("red", goal("red")))
    )
)
stage(
    sample("board", touch("green")),
    optimize(
        keep(touch('green')),
        require(similar("green", goal("green")))
    )
)


Please compile the input into a program. The generated programs
    satisfy several requirements: Do not use functions not mentioned
    before; the program should be concise; only use 'touch' at most
    once within a single 'optimize' and if we need to manipulate
    multiple objects, please use touch('all').

Please include an 'optimize' for each stage. Do not use more than one
    stage if it does not need to manipulate different objects or parts
    . Here is the input to compile:
```

B.1 Comparing GPT3.5 and GPT4

We notice that GPT4 has a better performance compared with GPT3.5. Below we provide the results of GPT3.5 and GPT4 of example tasks shown in the main text. Generally, we have found that it is common for GPT3.5 to misinterpret the instructions, add/ignore constraints, and write incorrect commands with incorrect grammar. In contrast, GPT4 can generate pretty good results.

Although GPT4 already outperforms GPT3.5 in the translation task, we observed several failure cases in the translation process. For example, the output of GPT4 will contain objects that do not exist. In a task that lifts a sphere above a box, the output may contain an invalid sentence like require(similar('sphere', goal('sphere above box'))) where 'sphere above box' which doesn't correspond to a proper object. It may also generate a statement with a type error. For example, sometimes it may output require(similar('sphere', y(max(pcd('box'))))), which would trigger a compilation error as we can not compute the shape distance between an object and a coordinate. Fortunately, those compilation errors could be detected during the compilation and can be resolved by resampling. However, there are also rare cases where LLM removes or adds more constraints to the instructions. For example, it may ask the solver to fix the position of an object that needs to move. In these cases, our solver may fail to find a suitable solution. Restricting the output to fit a particular syntax, for instance, using context-free grammars would resolve such problems.

Test Case 1 Grasp the front of the rope and then grasp the back of the rope. Deform the rope into its target shape.

GPT3.5: added constraints not mentioned in the language

```
rope = get("rope")
stage(
    sample("gripper", grasp(frontend(rope))),
    optimize(
        keep(touch(rope)),
        require(similar(rope, goal('rope'))),
        keep(no_break(rope))
    )
)
stage(
    sample("gripper", grasp(backpart(rope))),
    optimize(
        keep(touch(rope)),
        require(similar(rope, goal('rope'))),
        keep(no_break(rope))
    )
)
```

```
rope = get("rope")
stage(
    sample("gripper", grasp(frontend(rope))),
    optimize(
        keep(touch(frontend(rope))),
        require(similar(rope, goal("rope")), end=0.5)
    )
)
stage(
    sample("gripper", grasp(backend(rope))),
    optimize(
        keep(touch(backend(rope))),
        require(similar(rope, goal("rope")), end=1.0)
    )
)
```

Test case 2   Use the gripper to first lift up the white cube above the black wall and then move it to the target location.

GPT3.5: it creates two stages for single a stage task (not need to regrasp the object).

```
white_cube = get("white cube")
black_wall = get("black wall")
stage(
    sample("gripper", lift(white_cube)),
    optimize(
        require(gt(y(com(white_cube)), y(max(pcd(black_wall))))),
        keep(touch(white_cube)),
        keep(no_break(white_cube))
    )
)
stage(
    sample("gripper", gt(x(com(white_cube)), x(com(black_wall)))),
    optimize(
        require(l2(com(white_cube), com(goal("white cube")))),
        keep(touch(white_cube)),
        keep(no_break(white_cube))
    )
)
```

```
white_cube = get("white cube")
black_wall = get("black wall")

stage(
    sample("gripper", grasp(white_cube)),
    optimize(
        require(gt(y(com(white_cube)), y(max(pcd(black_wall)))), end
            =0.5),
        require(l2(com(white_cube), com(goal("white cube"))), end=1),
        keep(touch(white_cube))
    )
)
```

Test case 3   Put the knife above the white object. Cut the knife down to match the target shape of the white object. Ensure that the knife does not rotate.

GPT3.5: It uses z instead y to extract the objects height while the latter is true. It also misinterprets the meaning to deform the knife into the goal.

```
white_obj = get("white")
stage(
    sample("knife", gt(z(com("knife")), z(max(pcd(white_obj))))),
    optimize(
        require(l2(com("knife"), com(white_obj))),
        require(similar("knife", goal("white"))),
        keep(pitch() < 0.1, yaw() < 0.1)
    )
)
```

GPT4

```
stage(
    sample("knife", above(get("white"))),
    optimize(
        require(similar(get("white"), goal("white"))),
        keep(touch("white")),
        keep(roll() < 0.1),
        keep(pitch() < 0.1),
        keep(yaw() < 0.1)
    )
)
```

Test case 4   Pick up the front part of object 'top left mat' and place it into the goal without breaking it.

GPT3.5: It adds an incorrect command to let the gripper to move away from the object in the beginning.

```
mat = get("top left mat")
stage(
    sample("gripper", grasp(frontpart(mat)), away()),
    optimize(
        require(similar(mat, goal('top left mat'))),
        keep(touch(mat)),
        keep(no_break(mat))
    )
)
```

```
stage(
    sample("gripper", grasp(frontpart(get("top left mat")))),
    optimize(
        require(l2(com(get("top left mat")), com(goal("top left mat"))
            )),
        keep(touch(frontpart(get("top left mat")))),
        keep(no_break(get("top left mat")))
    )
)
```

C   Reinforcement learning baseline

We use the SAC and PPO implementation from stable-baselines3 with hyperparameters in Table 3. Figure 8 compares the optimization performance of RL algorithms and the Adam optimizer with respect to the IOU score of the ending state with different numbers of sampled episodes for the 20 single-stage tasks of five categories.

Table 3: Parameters for Reinforcement Learning

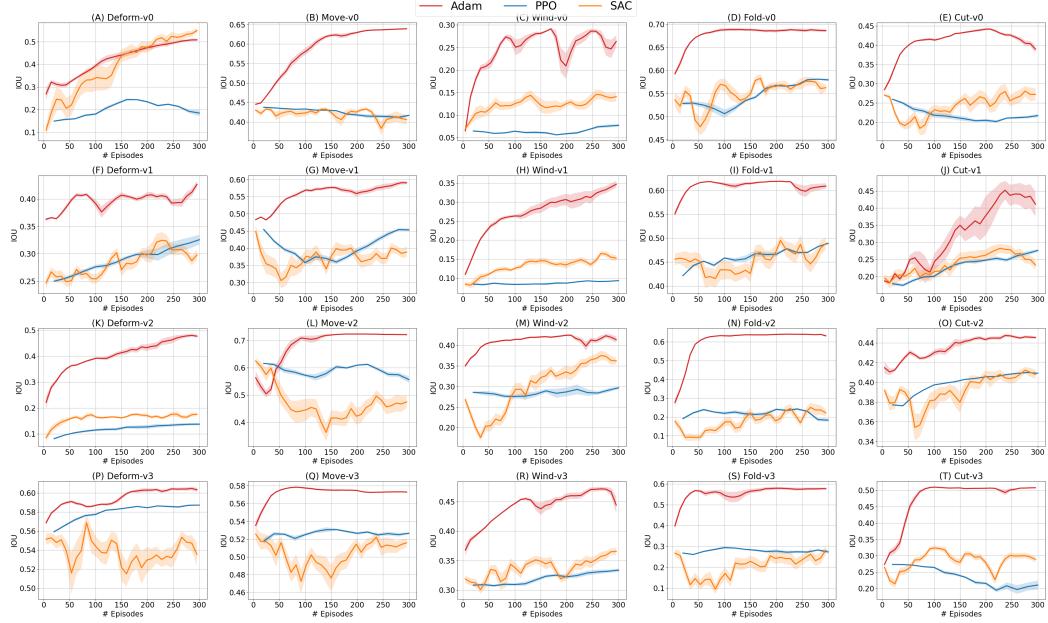| Table 4: SAC		Table 5: PPO Parameters	
gamma	0.95	update steps	2048
learning rate	3×10^{-4}	learning rate	3×10^{-4}
buffer size	10^6	entropy coef	0
target update coef	0.005	value loss coef	0.5
batch size	256	batch size	64

Figure 8: Comparison between RL and differentiable physics solver

D   Evaluation metrics

We notice that in some tasks, a simple Intersection Over Union (IOU) score comparing shapes does not effectively explain the human intuition of task completion. For example, a solver might "cut" two

objects similar to the target shape without truly separating them, as the space between the two parts might only contain a minimal amount of space. In other circumstances, the solver may violate the requirements of no_break and break a rope into two or destroy a mat. To this end, we add checkers to check if two shapes are split and if a shape is not broken for success evaluation. For no-breaking constraint, we track particles and ensure their distance to their initial nearest neighbor does not change too much. On the other hand, we can check if two shapes are separated from each other by finding the connected components of particles.