

# LLMPHY: COMPLEX PHYSICAL REASONING USING LARGE LANGUAGE MODELS AND WORLD MODELS

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Physical reasoning is an important skill needed for robotic agents when operating in the real world. However, solving such reasoning problems often involves hypothesizing and reflecting over complex multi-body interactions under the effect of a multitude of physical forces and thus learning all such interactions poses a significant hurdle for state-of-the-art machine learning frameworks, including large language models (LLMs). To study this problem, we propose a new physical reasoning task and a dataset, dubbed *TraySim*. Our task involves predicting the dynamics of several objects on a tray that is given an external impact – the domino effect of the ensued object interactions and their dynamics thus offering a challenging yet controlled setup, with the goal of reasoning being to infer the stability of the objects after the impact. To solve this complex physical reasoning task, we present LLMPHy, a zero-shot black-box optimization framework that leverages the physics knowledge and program synthesis abilities of LLMs, and synergizes these abilities with the world models built into modern physics engines. Specifically, LLMPHy uses an LLM to generate code to iteratively estimate the physical hyperparameters of the system (friction, damping, layout, etc.) via an implicit analysis-by-synthesis approach using a (non-differentiable) simulator in the loop and uses the inferred parameters to imagine the dynamics of the scene towards solving the reasoning task. To show the effectiveness of LLMPHy, we present experiments on our TraySim dataset to predict the steady-state poses of the objects. Our results show that the combination of the LLM and the physics engine leads to state-of-the-art zero-shot physical reasoning performance, while demonstrating superior convergence against standard black-box optimization methods and better estimation of the physical parameters. Further, we show that LLMPHy is capable of solving both continuous and discrete black-box optimization problems.

## 1 INTRODUCTION

Many recent Large Language models (LLMs) appear to demonstrate the capacity to effectively capture knowledge from vast amounts of multimodal training data and their generative capabilities allow humans to naturally interact with them towards extracting this knowledge for solving challenging real-world problems. This powerful paradigm of LLM-powered problem solving has manifested in a dramatic shift in the manner of scientific pursuit towards modeling research problems attuned to a form that can leverage this condensed knowledge of the LLMs. A few notable such efforts include, but not limited to the use of LLMs for robotic planning (Song et al., 2023; Kim et al., 2024), complex code generation (Tang et al., 2024; Jin et al., 2023), solving optimization problems (Yang et al., 2024; Hao et al., 2024), conduct sophisticated mathematical reasoning (Trinh et al., 2024), or even making scientific discoveries (Romera-Paredes et al., 2024).

While current LLMs seem to possess the knowledge of the physical world and may be able to provide a plan for solving a physical reasoning task (Singh et al., 2023; Kim et al., 2024) when crafted in a suitable multimodal format (prompt), their inability to interact with the real-world or measure unobservable attributes of the world model, hinders their capacity in solving complex physical reasoning problems (Wang et al., 2023; Bakhtin et al., 2019; Riochet et al., 2021; Harter et al., 2020; Xue et al., 2021). Consider for example the scene in Figure 1, where the LLM is provided as input the first image and is asked to answer: *which of the objects will remain standing on the tray when impacted by the pusher if the pusher collides with the tray with a velocity of 4.8 m/s?*. To answer this

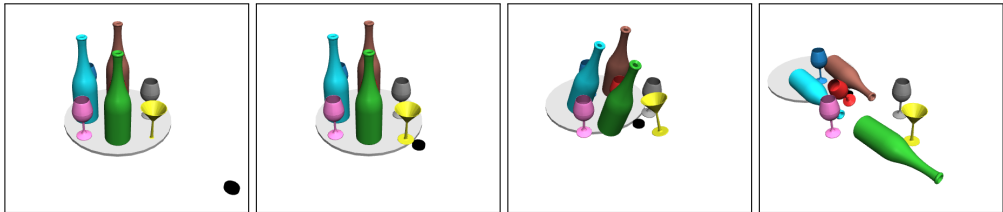


Figure 1: Frames from an example dynamical sequence in our TraySim dataset. The left-most frame shows the first frame of the scene with many objects on the tray and is going to be impacted by a black pusher (right-bottom). The subsequent frames show the state of the system at the 25-th, 50-th, and the 200-th time step (each step is 0.01s). Our task is for the LLM to reason through the dynamics of the system and predict the stability of each object on the tray at the end of the episode, in a zero-shot manner.

question, the LLM must know the various physical attributes of the system, including the masses, friction coefficients, and forces, among others. While, a sophisticated LLM may be able to give an educated guess based on the intuitive physics of the system extracted from its training data, a useful solution would demand a more intricate reasoning path in estimating the real-world physics and dynamics of the given system; such complex dynamics may be difficult or even impossible to be learned solely from training data. Conversely, advancements in graphics hardware and software have led to the development of advanced physics engines capable of simulating realistic world models. Thus, rather than having the LLM to learn the world physics, our key idea is to consider using a physics engine in tandem with the LLM, where the LLM may use its world knowledge for generating scene-based reasoning hypotheses while the simulator is used to verify them within the physical world model.

To study this problem, we consider the novel task of predicting the dynamics of objects and their stability under the influence of an impact – an important problem for a variety of robotic applications (Gasparetto et al., 2015; Ahmed et al., 2020). In this paper, we consider this problem in a challenging setting using our new dataset, *TraySim*, in which the impact is caused by a pusher colliding to a tray that holds several objects of varied sizes, masses, and centers of gravity, with the goal of predicting the dynamics of each of the object instances. We cast this task as that of answering physical reasoning questions. Specifically, as illustrated in Figure 1, *TraySim* includes simulated video sequences consisting of a tray with an arbitrary number of objects on it and given the first video frame of a given scene, the task of the reasoning model is to infer which of the objects on the tray will remain upright after the impact when the system has stabilized. As is clear from Figure 1, solving this task will require the model to derive details regarding the physical properties of each of the objects and their contacts, as well as have the ability to imagine the system’s dynamics through multi-body interactions influenced by the various internal and external forces from the impact. Our task presents a challenging reasoning setup for current machine learning models, including LLMs.

To solve this task, we propose LLMPhy, a [black-box optimization setup combining an LLM with a physics engine that leverages the program synthesis abilities of the LLM to communicate with the engine for solving our task](#). LLMPhy operates in two phases: i) a parameter estimation phase, where LLMPhy is used as a [continuous black-box optimization module towards inferring the physical parameters of the objects](#), including the friction, stiffness, damping, etc. from a given example video sequence, and ii) a scene understanding phase, where the LLM-simulator combination is used as a [discrete black-box optimizer to reconstruct the problem layout for synthesizing the setup within the simulator for execution](#). Our framework builds a feedback loop between the LLM and the physics engine, where the LLM generates programs using its estimates of physical attributes; the programs are executed in the simulator, and the error from the simulations are fed back to the LLM as prompts to refine its estimates until a suitable convergence criteria is met. Note that we do not assume any differentiability properties of the simulator, which makes our setup highly general. This allows the approach to function as a black-box optimization framework, enabling its use with a wide range of simulators without the need for gradient-based methods.

While we may generate unlimited data using our simulation program, given the zero-shot nature of our setup, we synthesized 100 sequences in our *TraySim* dataset to demonstrate the effectiveness of

LLMPhy. Each sample in TraySim has two video sequences: i) the task sequence of which only the first frame is given to a reasoning agent, and ii) a parameter-estimation video sequence which has a lesser number of instances of each of the object types appearing in the task sequence; the latter sequence has an entirely different layout and dynamics of objects after its specific impact settings. To objectively evaluate performance, we cast the task as physical question answering problem, where the LLM is required to select the correct subset of answers from the given candidate answers. Our results on TraySim show that LLMPhy leads to clear improvements in performance ( $\sim 3\%$  accuracy) against alternatives on the QA task, including using Bayesian optimization, CMA-ES, and solely using an LLM for physical reasoning, while demonstrating better convergence and estimation of the physical parameters.

Before moving forward, we summarize below our main contributions:

- We consider the novel task of reasoning over complex physics of a highly dynamical system by combining LLMs with possibly non-differentiable physics engines.
- We propose a zero-shot reasoning framework LLMPhy, which combines the reasoning and program synthesis abilities of an LLM with the realistic simulation abilities of a physics engine. This approach is used to estimate the physical parameters of the model, the scene layout, and synthesizing the dynamical scene for inferring the solution.
- We introduce a novel synthetic multi-view dataset: TraySim, to study this task. The dataset consists of 100 scenes for zero-shot evaluation.
- Our experiments demonstrate state-of-the-art performances using LLMPhy highlighting its potential for tackling complex physics-based tasks involving both discrete and continuous optimization sub-tasks.

## 2 RELATED WORKS

Large language models (LLMs) demonstrate remarkable reasoning skills across a variety of domains, highlighting their versatility and adaptability. They have shown proficiency in managing complex conversations (Glaese et al., 2022; Thoppilan et al., 2022), engaging in methodical reasoning processes (Wei et al., 2022; Kojima et al., 2022), planning (Huang et al., 2022), tackling mathematical challenges (Lewkowycz et al., 2022; Polu et al., 2022), and even generating code to solve problems (Chen et al., 2021). As we start to incorporate LLMs into physically embodied systems, it’s crucial to thoroughly assess their ability for physical reasoning. However, there has been limited investigation into the physical reasoning capabilities of LLMs.

In the field of language-based physical reasoning, previous research has mainly concentrated on grasping physical concepts and the attributes of different objects. (Zellers et al., 2018) introduced grounded commonsense inference, merging natural language inference with commonsense reasoning. Meanwhile, (Bisk et al., 2020) developed the task of physical commonsense reasoning and a corresponding benchmark dataset, discovering that pretrained models often lack an understanding of fundamental physical properties. (Aroca-Ouellette et al., 2021) introduced a probing dataset that evaluates physical reasoning through multiple-choice questions. This dataset tests both causal and masked language models in a zero-shot context. However, many leading pretrained models struggle with reasoning about physical interactions, particularly when answer choices are reordered or questions are rephrased. (Tian et al., 2023) explored creative problem-solving capabilities of modern LLMs in constrained setting. They automatically generate a dataset consisting of real-world problems deliberately designed to trigger innovative usage of objects and necessitate out-of-the-box thinking. (Wang et al., 2023) presented a benchmark designed to assess the physics reasoning skills of large language models (LLMs). It features a range of object-attribute pairs and questions aimed at evaluating the physical reasoning capabilities of various mainstream language models across foundational, explicit, and implicit reasoning tasks. The results indicate that while models like GPT-4 demonstrate strong reasoning abilities in scenario-based tasks, they are less consistent in object-attribute reasoning compared to human performance.

In addition to harnessing LLMs for physical reasoning, recent works have used LLMs for optimization. The main focus has been on targeted optimization for employing LLMs to produce prompts that improves performance of another LLM. (Yang et al., 2024) shows that LLMs are able to find good-quality solutions simply through prompting on small-scale optimization problems. They demon-

strate the ability of LLMs to optimize prompts where the goal is to find a prompt that maximizes the task accuracy. The applicability of various optimization methods depends on whether the directional feedback information is available. In cases when the directional feedback is available, one can choose efficient gradient-based optimization methods (Sun et al., 2019). However, in scenarios without directional feedback, black-box optimization methods (Terayama et al., 2021) are useful such as Bayesian optimization (Mockus, 1974), Multi-Objective BO (Konakovic Lukovic et al., 2020) and CMA-ES (Hansen & Ostermeier, 2001). Only a limited number of studies have explored the potential of LLMs for general optimization problems. (Guo et al., 2023) shows that LLMs gradually produce new solutions for optimizing an objective function, with their pretrained knowledge significantly influencing their optimization abilities. (Nie et al., 2024) study factors that make an optimization process challenging in navigating a complex loss function. They conclude that LLM-based optimizer’s performance varies with the type of information the feedback carries, and given proper feedback, LLMs can strategically improve over past outputs. In contrast to these prior works, our goal in this work is to combine an LLM with a physics engine for physics based optimization.

Our work is inspired by the early work in neural de-rendering (Wu et al., 2017) that either (re-) simulates a scene using a physics engine or synthesizes realistic scenes for physical understanding Bear et al. (2021). Similar to our problem setup, CoPhy Baradel et al. (2019) and ComPhy Chen et al. (2022) consider related physical reasoning tasks, however with simplistic physics and using supervised learning. In (Liu et al., 2022), a language model is used to transform a given reasoning question into a program for a simulator, however does not use the LLM-simulator optimization loop as in LLMPhy. In SimLM (Memery et al., 2023), an LLM-simulator combination is presented for predicting the physical parameters of a projectile motion where the feedback from a simulator is used to improve the physics estimation in an LLM, however assumes access to in-context examples from previous successful runs for LLM guidance. In Eureka Ma et al. (2023), an LLM-based program synthesis is presented for designing reward functions in a reinforcement learning (RL) setting, where each iteration of their evolutionary search procedure produces a set of LLM generated candidate reward functions. Apart from the task setup, LLMPhy differs from Eureka in two aspects: (i) Eureka involves additional RL training that may bring in training noise in fitness evaluation, (ii) does not use full trajectory of optimization in its feedback and as a result, the LLM may reconsider previous choices. See F for details.

### 3 PROPOSED METHOD

The purpose of this work is to enable LLMs to perform physics-based reasoning in a zero-shot manner. Although LLMs may possess knowledge of physical principles that are learned from their training data, state-of-the-art models struggle to effectively apply this knowledge when solving specific problems. This limitation, we believe, is due to the inability of the model to interact with the scene to estimate its physical parameters, which are essential and needs to be used in the physics models for reasoning, apart from the stochastic attributes implicit in any such system. While, an LLM may be trained to implicitly model the physics given a visual scene – e.g., generative models such as SoRA<sup>1</sup>, Emu-video Girdhar et al. (2023), etc., may be considered as world model simulators – training such models for given scenes may demand exorbitant training data and compute cycles. Instead, in this paper, we seek an alternative approach by leveraging the recent advancements in realistic physics simulation engines and use such simulators as a tool accessible to the LLM for solving its given physical reasoning task. Specifically, we attempt to solve the reasoning task as that of equipping the LLM to model and solve the problem using the simulator, and for which we leverage on the LLM’s code generation ability as a bridge. In the following sections, we expisit the technical details involved in achieving this LLM-physics engine synergy.

#### 3.1 PROBLEM SETUP

Suppose  $\mathbf{X}^v = \langle \mathbf{x}_1^v, \mathbf{x}_2^v, \dots, \mathbf{x}_T^v \rangle$  denote a video sequence with  $T$  frames capturing the dynamics of a system from a camera viewpoint  $v$ . We will omit the superscript  $v$  when referring to all the views jointly. In our setup, we assume the scene consists of a circular disk (let us call it a *tray*) of a given radius, friction, and mass. Further, let  $\mathcal{C}$  denote a set of object types, e.g., in Figure 1, there are three types of objects: a *bottle*, a *martini glass*, and a *wine glass*. The tray is assumed to hold

<sup>1</sup><https://openai.com/index/sora/>

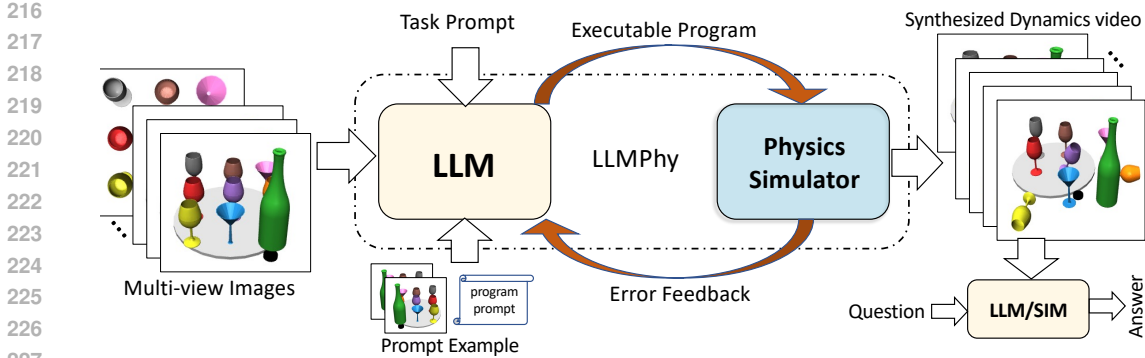


Figure 2: Illustration of the key components of LLMPhy and the control flow between LLM, physics simulator, and the varied input modalities and examples.

a maximum of  $K$  object instances, the  $k$ -th instance is denoted  $o_k$ ;  $K$  being a perfect square. To simplify our setup, we assume that the instances on the tray are arranged on a  $\sqrt{K} \times \sqrt{K}$  regular grid, with potentially empty locations. We further assume that the masses of the objects in  $\mathcal{C}$  are given during inference, while other physical attributes, denoted as  $\Phi_c$  for all objects  $c \in \mathcal{C}$ , are *unknown* and identical for objects of the same type. In line with the standard Mass-Spring-Damping (MSD) dynamical system, we consider the following set of contact physics parameters  $\Phi_c \in \mathbb{R}^4$  for each object class: i) coefficient of sliding friction, ii) stiffness, iii) damping, and iv) the rotational inertia (also called armature). To be clear, we do not assume or use any physics model in our optimization pipeline, and our setup is entirely black-box, but the selection of these optimization physics parameters is inspired by the MSD model. We assume the objects do not have any rotational or spinning friction. While the instances  $o_k$  of the same type are assumed to share the same physics parameters, they differ in their visual attributes such as color or shape. The tray is impacted by a pusher  $p$  that starts at a fixed location and is given an initial velocity of  $p_s$  towards the tray. The pusher is assumed to have a fixed mass and known physical attributes, and the direction of impact is assumed to coincide with the center of the circular tray.

### 3.2 PROBLEM FORMULATION

With the notation above, we are now ready to formally state our problem. In our setup, we define an input task instance as:  $\mathcal{T} = (\{\mathbf{x}_g^v\}_{v \in |\mathcal{V}|}, p_s, Q, \mathcal{O}, \mathcal{I}, \mathbf{X}_{\mathcal{T}}, \mathcal{C}_{\mathcal{T}})$ , where  $\mathbf{x}_g$  is the first frame of a video sequence  $\mathbf{X}$  with  $\mathcal{V}$  views,  $p_s$  is the initial velocity of the pusher  $p$ ,  $Q$  is a question text describing the task, and  $\mathcal{O}$  is a set of answer candidates for the question. The goal of our reasoning agent is to select the correct answer set  $\mathcal{A} \subset \mathcal{O}$ . The notation  $\mathcal{C}_{\mathcal{T}} \subseteq \mathcal{C}$  denotes the subset of object classes that are used in the given task example  $\mathcal{T}$ . In this paper, we assume the question is the same for all task examples, i.e., *which of the object instances on the tray will remain steady when impacted by the pusher with a velocity of  $p_s$ ?* We also assume to have been given a few in-context examples  $\mathcal{I}$  that familiarizes the LLM on the structure of the programs it should generate. We found that such examples embedded in the prompt are essential for the LLM to restrict its generative skills to the problem at hand, while we emphasize that the knowledge of these in-context examples will not by themselves help the LLM to correctly solve a given test example.

As it is physically unrealistic to solve the above setup using only a single image (or multiple views of the same time-step), especially when different task examples have distinct dynamical physics parameters  $\Phi$  for  $\mathcal{C}_{\mathcal{T}}$ , we also assume to have access to an additional video sequence  $\mathbf{X}_{\mathcal{T}}$  associated with the given task example  $\mathcal{T}$  containing the same set of objects as in  $\mathbf{x}_g$  but in a different layout and potentially containing a smaller number of object instances. **The purpose of having  $\mathbf{X}_{\mathcal{T}}$  is to estimate the physics parameters of the objects in  $\mathbf{x}_g$ , so that these parameters can then be used to conduct physical reasoning for solving  $\mathcal{T}$ , similar to the setup in Baradel et al. (2019); Chen et al. (2022).** Note that this setup closely mirrors how humans would solve such a reasoning task. Indeed, humans may pick up and interact with some object instances in the scene to understand their physical properties, before applying sophisticated reasoning on a complex setup. Without any loss



of generality, we assume the pusher velocity in  $\mathbf{X}_{\mathcal{T}}$  is fixed across all such auxiliary sequences and is different from  $p_s$ , which varies across examples.

### 3.3 COMBINING LLMs AND PHYSICS ENGINES FOR PHYSICAL REASONING

In this section, our proposed LLMPHy method for our solving physical reasoning task is outlined. Figure 2 illustrates our setup. Since LLMs on their own may be incapable of performing physical reasoning over a given task example, we propose combining the LLM with a physics engine. The physics engine provides the constraints of the world model and evaluates the feasibility of the reasoning hypothesis generated by the LLMs. This setup provides feedback to the LLM that enables it reflect on and improve its reasoning. Effectively solving our proposed task demands inferring two key entities: i) the physical parameters of the setup, and ii) layout of the task scene for simulation using physics to solve the task. We solve for each of these sub-tasks in two distinct phases as detailed below. Figure 3 illustrates our detailed architecture, depicting the two phases and their interactions.

#### 3.3.1 LLMPHy PHASE 1: INFERRING PHYSICAL PARAMETERS

As described above, given the task example  $\mathcal{T}$ , LLMPHy uses the task video  $\mathbf{X}_{\mathcal{T}}$  to infer the physical attributes  $\Phi$  of the object classes in  $\mathcal{C}$ . Note that these physical attributes are specific to each task example. Suppose  $\tau : \mathcal{X} \rightarrow \mathbb{R}^{3 \times T \times |\mathcal{C}_{\mathcal{T}}|}$  be a function that extracts the physical trajectories of each of the objects in the given video  $\mathbf{X}_{\mathcal{T}} \in \mathcal{X}$ , where  $\mathcal{X}$  denotes the set of all videos.<sup>2</sup> Note that we have used a subscript of  $\mathcal{T}$  with  $\mathcal{C}$  to explicitly show the subset of object types that may be appearing in the given task example.

Suppose  $\text{LLM}_1$  denotes the LLM used in phase 1<sup>3</sup>, which takes as input the in-context examples  $\mathcal{I}_1 \subset \mathcal{I}$  and the object trajectories from  $\mathbf{X}_{\mathcal{T}}$ , and is tasked to produce a program  $\pi(\Phi) \in \Pi$ , where  $\Pi$  denotes the set of all programs. Further, let  $\text{SIM} : \Pi \rightarrow \mathbb{R}^{3 \times T \times \mathcal{C}_{\mathcal{T}}}$  be a physics-based simulator that takes as input a program  $\pi(\Phi) \in \Pi$  and produce trajectories of objects described by the program using the physics attributes. Then, the objective of phase 1 of  $\text{LLMPHy}_1$  can be described as:

$$\arg \min_{\Phi} \|\text{LLMPHy}_1(\pi(\Phi) \mid \tau(\mathbf{X}_{\mathcal{T}}), \mathcal{I}_1) - \tau(\mathbf{X}_{\mathcal{T}})\|^2, \tag{1}$$

where  $\text{LLMPHy}_1 = \text{SIM} \circ \text{LLM}_1$  is the composition of the simulator and the LLM through the generated program, with the goal of estimating the correct physical attributes of the system  $\Phi$ . Note that the notation  $\pi(\Phi)$  means the generated program takes as argument the physics parameters  $\Phi$  which is what we desire to optimize using the LLM.

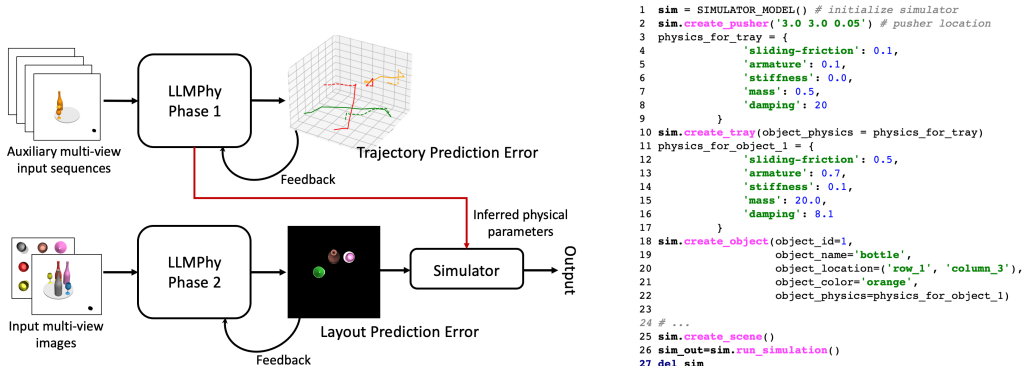


Figure 3: Left: Full architecture of the two phases in LLMPHy. Right: A simplified LLMPHy program. We abstract the complexity in running the simulations through simple API calls so that LLM can focus on the optimization variables. See Appendix I for full program examples.

<sup>2</sup>In experiments, we use the simulator to extract object trajectories, thus implementing  $\tau$ . See Appendix D.1.

<sup>3</sup>We use the same LLM in both phases, but the notation is only for mathematical precision.

### 3.3.2 LLMPHy PHASE 2: SIMULATING TASK EXAMPLE

The second phase of LLMPHy involves applying the inferred physical parameters  $\Phi$  for the object classes in  $\mathcal{C}$  to solve the task problem described in  $\mathbf{x}_g$ , i.e., the original multi-view task images (see Figure 3). This involves solving a perception task consisting of two steps: i) understanding the scene layout (i.e., where the various object instances are located on the tray, their classes, and attributes (e.g., color); this is important as we assume that different type of objects have distinct physical attributes, and ii) using the physical attributes and the object layout to produce code that can be executed in the physics engine to simulate the full dynamics of the system to infer the outcome; i.e., our idea is to use the simulator to synthesize a dynamical task video from the given input task images, and use the ending frames of this synthesized video to infer the outcome (see Figure 2).

Suppose  $\text{LLM}_2$  denotes the LLM used in Phase 2, which takes as input the multi-view task images  $\mathbf{x}_g$ , the physical attributes  $\Phi^*$ , and Phase 2 in-context examples  $\mathcal{I}_2 \subset \mathcal{I}$  to produce a program  $\pi(\Psi) \in \Pi$  that reproduces the scene layout parameters, i.e., the triplet  $\Psi = \{(\text{class}, \text{location}, \text{color})\}_k$  for each instance. The objective for estimating the layout parameters  $\Psi$  can be written as:

$$\Psi^* = \arg \min_{\Psi} \|\text{LLMPHy}_2(\pi(\Psi) \mid \mathbf{x}_g, \mathcal{I}_2) - \mathbf{x}_g\|^2, \quad (2)$$

where  $\text{LLMPHy}_2 = \text{SIM} \circ \text{LLM}_2$ . Once the correct layout parameters  $\Psi^*$  are estimated, we can produce a video sequence  $\hat{\mathbf{X}} \mid \Psi^*, \Phi^*$  using the simulator, and which can then be used for solving the problem by selecting an answer subset  $\mathcal{A}$  from the answer options  $\mathcal{O}$ . [We may use an LLM or extract the pose of the instances within the simulator to solve the question-answering task; in this work, we use the latter for convenience.](#)

### 3.4 OPTIMIZING LLM-SIMULATOR COMBINATION

In Alg. 1, we detail the steps for optimizing LLMPHy. Given that we assume the simulator might be non-differentiable, we frame this as a black-box optimization problem. Here, the optimization variables are sampled based on the inductive bias and the knowledge of physics learned by the LLM from its large corpora of training data. The LLM generates samples over multiple trials, which are then validated using the simulator. The resulting error is used to refine the LLM’s hyper-parameter search. A key insight of our approach is that, since the hyper-parameters in our setup have physical interpretations in the real-world, a knowledgeable LLM should be capable of selecting them appropriately by considering the error produced by its previous choices. In order for the LLM to know the history of its previous choices and the corresponding error induced, we augment the LLM prompt with this optimization trace from the simulator at each step.

---

**Algorithm 1** Pseudo-code describing the key steps in optimizing LLMPHy for phases 1 and 2.

---

**Require:**  $\mathbf{X}, \Lambda$  ▷  $\mathbf{X}$  is the input data, and  $\Lambda$  is the desired result, e.g., trajectory, layout, etc.  
 prompt  $\leftarrow$  'task prompt' ▷ We assume here a suitable prompt for the LLM.  
**for**  $i = 1$  to max\_steps **do**  
    $\pi \leftarrow \text{LLM}(\mathbf{X}, \mathcal{I}, \text{prompt})$  ▷ Generated program  $\pi$  is assumed to have the optimization variables.  
    $\hat{\Lambda} \leftarrow \text{SIM}(\pi)$  ▷ SIM reproduced result from  $\pi$ .  
   error  $\leftarrow \|\Lambda - \hat{\Lambda}\|^2$   
   **if** error  $\leq \epsilon$  **then**  
     **return**  $\pi$   
   **else**  
     prompt  $\leftarrow \text{concat}(\text{prompt}, \pi, \text{concat}(\text{"Error ="}, \text{error}))$   
   **end if**  
**end for**

---

## 4 EXPERIMENTS AND RESULTS

In this section, we detail our simulation setup used to build our TraySim dataset, followed by details of other parts of our framework, before presenting our results.

**Simulation Setup:** As described above, we determine the physical characteristics of our simulation using a physics engine. MuJoCo Todorov et al. (2012) was used to setup the simulation and compute the rigid body interactions within the scene. It is important to note that any physics engine capable of computing the forward dynamics of a multi-body system can be integrated within our framework as the simulation is exposed to the LLM through Python API calls for which the physical parameters and layout are arguments. As a result, the entirety of the simulator details are abstracted out from the LLM. Our simulation environment is built upon a template of the *World*, which contains the initial parametrization of our model of Newtonian physics. This includes the gravity vector  $\mathbf{g}$ , time step, and contact formulation, but also graphical and rendering parameters later invoked by the LLM when executing the synthesized program. See Appendix A for details.

**TraySim Dataset:** Using the above setup, we created 100 task sequences using object classes  $\mathcal{C} = \{\text{wine glass, martini glass, bottle}\}$  with object instances from these classes arranged roughly in a  $3 \times 3$  matrix on the tray. The instance classes and the number of instances are randomly chosen with a minimum of 5 and a maximum of 9. Each task sequence is associated with an auxiliary sequence for parameter estimation that contains at least one object instance from every class of object appearing in the task images. We assume each instance is defined by a triplet: (color, type, location), where the color is unique across all the instances on the tray so that it can be identified across the multi-view images. The physical parameters of the objects are assumed to be the same for both the task sequences and the auxiliary sequences, and instances of the same object classes have the same physical parameters. The physics parameters were randomly sampled for each problem in the dataset. Each sequence was rendered using the simulator for 200 time steps, each step has a duration of 0.01s. We used the last video frame from the task sequence to check the stability of each instance using the simulator. We randomly select five object instances and create a multiple choice candidate answer set for the question-answering task, where the ground truth answer is the subset of the candidates that are deemed upright in the last frame. In Figure 4, we illustrate the experimental setup using an example from the TraySim dataset. See Appendix B for more details of the physics parameters, and other settings.

**Large Language Model:** We use the OpenAI o1-mini text-based LLM for our Phase 1 experiments and GPT-4o vision-and-language model (VLM) in Phase 2. Recall that in Phase 1 we pre-extract the object trajectories for optimization.

**Phase 1 Details:** In this phase, we provide as input to the LLM four items: i) a prompt describing the problem setup, the qualitative parameters of the objects (such as mass, height, size of tray, etc.) and the task description, ii) an in-context example consisting of sample trajectories of the object instances from its example auxiliary sequence, iii) a program example that, for the given example auxiliary sequence trajectories, shows their physical parameters and the output structure, and iv) auxiliary task sequence trajectories (from the sequence for which the physical parameters have to be estimated) and a prompt describing what the LLM should do. The in-context example is meant to guide the LLM to understand the setup we have, the program structure we expect the LLM to synthesize, and our specific APIs that need to be called from the synthesized program to reconstruct the scene in our simulator. Please see our Appendices D and I for details.

**Phase 2 Details:** The goal of the LLM in Phase 2 is to predict the object instance triplet from the multi-view task images. Towards this end, the LLM generates code that incorporates these triplets, so that when this code is executed, the simulator will reproduce the scene layout. Similar to Phase 1, we provide to the LLM an in-context example for guiding its code generation, where this in-context example contains multi-view images and the respective program, with the goal that the LLM learns the relation between parts of the code and the respective multi-view images, and use this knowledge to write code to synthesize the layout of the provided task images. When iterating over the optimization steps, we compute an error feedback to the LLM to improve its previously generated code. See Appendix D and I for precise details on the feedback.

**LLMPhy Feedback Settings:** We compute the trajectory reconstruction error in Phase 1 where the synthesized program from the LLM containing the estimated physics parameters is executed in the simulator to produce the motion trajectory of the center of gravity of the instances. We sample the trajectory for every 10 steps and compute the L2 norm between the input and reconstructed trajectories. We use a maximum of 30 LLM-simulator iterations in Phase 1 and use the best reconstruction error to extract the parameters. For Phase 2, we use the Peak Signal-to-Noise ratio (PSNR) in the reconstruction of the first frame by the simulator using the instance triplets predicted by the LLM in



Expt #	Phase 1	Phase 2	mIoU (%)		LLMPhy	LLMPhy (1 iter.)
1	Random	Random	19.0			
2	N/A	LLM	32.1	C+L (%)	68.7	50.0
3	Random	LLMPhy	50.8	L+T (%)	66.3	49.3
4	BO	LLMPhy	59.6	C+L+T (%)	56.0	36.8
5	CMA-ES	LLMPhy	59.7			
6	LLMPhy	LLMPhy	<b>62.0</b>			
7	GT	LLMPhy	65.1			
8	CMA-ES	GT	75.8			
9	LLMPhy	GT	77.5			

Table 2: Experiments presenting the accuracy of generated code compared to the ground truth in Phase 2 of LLMPhy. We report the accuracy of matching the color (C) of the objects, their locations (L) on the 3 × 3 grid, and their type (T).

Table 1: Performances on TraySim QA task.

the generated program. We used a maximum of 5 LLMPhy iterations for this phase. As the LLM queries are expensive, we stopped the iterations when the trajectory prediction error is below 0.1 on average for Phase 1 and when the PSNR is more than 45 dB for Phase 2.

**Evaluation Metric and Baselines:** We consider various types of evaluations in our setup. Specifically, we use the intersection-over-union as our key performance metric that computes the overlap between the sets of LLMPhy produced answers in Phase 2 with the ground truth answer set. We also report the performances for correctly localizing the instances on the tray, which is essential for simulating the correct scene. As ours is a new task and there are no previous approaches that use the composition of LLM and physics engine, we compare our method to approaches that are standard benchmarks for continuous black-box optimization, namely using Bayesian optimization Mockus (1974) and Covariance matrix adaptation evolution strategy (CMA-ES) Hansen & Ostermeier (2001); Hansen (2016).

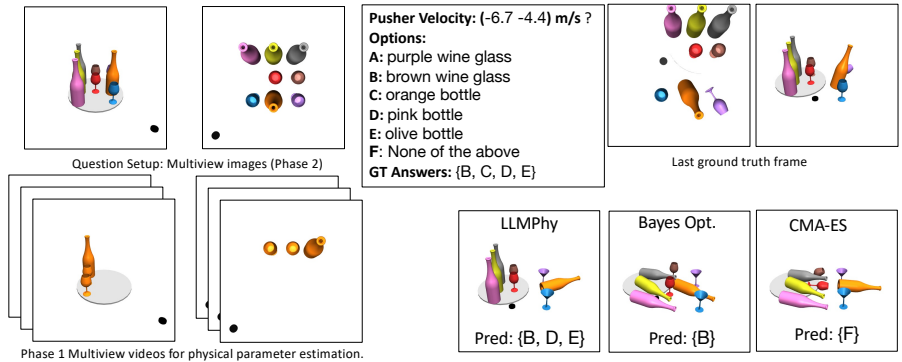


Figure 4: A sample qualitative result using LLMPhy, BO, and CMA-ES illustrating our problem setup. We omit the task question, which is the same for all problems, except the pusher velocity.

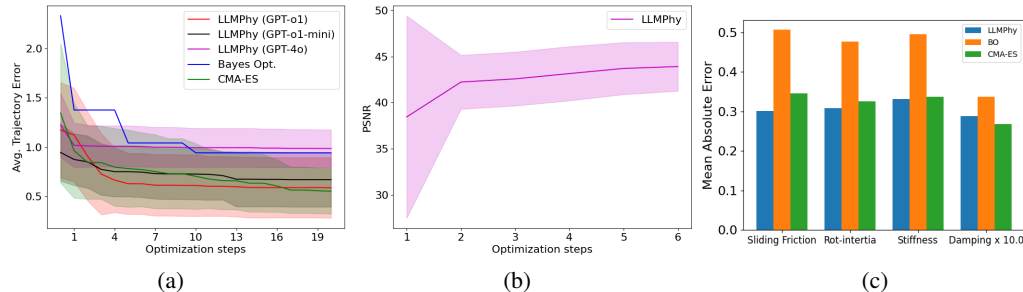


Figure 5: (a) Convergence comparisons using state-of-the-art LLMs in LLMPhy against Bayesian optimization and CMA-ES. We plot the *minimum loss computed thus far* in the optimization against the number of optimization steps. (b) shows the convergence of LLMPhy in Phase 2. (c) Comparison of physical parameter estimation error against alternatives using the ground truth.

**Comparisons to Prior Methods:** In Table 1, we compare the performance of Phase 1 and Phase 2 of LLMPHy to various alternatives and prior black-box optimization methods. Specifically, we see that random parameter sampling (Expt. #1) for the two phases lead to only 20% accuracy. Next, in Expt. #2, we use the Phase 2 multiview images (no sequence) and directly ask the GPT-4o to predict the outcome of the interaction (using the ground truth physics parameters provided), this leads to 32% accuracy, suggesting the LLM may provide an educated guess based on the provided task images. In Expt. #3, we use LLMPHy for Phase 2, however use random sampling for the physics parameters. We see that this leads to some improvement in performance, given we are using the simulator to synthesize the dynamical scene. Although the performance is lower than ideal and as noted from Figure 6 in the Appendix, we see that the outcome is strongly dependent on the physics parameters. In Expt. #4 and #5, we compare to prior black-box optimization methods for estimating the physics parameters while keeping the Phase 2 inference from LLMPHy as in the Expt. #3. To be comparable, we used 30 iterations for all methods.<sup>4</sup> As can be noted from the table, LLMPHy leads to about 2.3% better QA accuracy as is seen in Expt. #6. In Expt #7, we used the ground truth (GT) physics attributes for the respective objects in the simulation, and found 65.1% accuracy, which forms an upper-bound on the accuracy achievable from Phase 1. In Expt. #8 and #9, we compare the performance using GT phase 2 layout. We find from the performances that the physics parameters produced by LLMPHy are better than CMA-ES. In Table 2, we present the accuracy of LLMPHy in localizing the triplets in Phase 2. We find that with nearly 56% accuracy, LLMPHy estimates all the triplets and the performance improves over LLMPHy iterations. See detailed experiments and ablation studies in Appendix E.

**Convergence and Correctness of Physical Parameters:** In Figure 10(a), we plot the mean convergence (over a subset of the dataset) when using GPT-4o, o1-mini, Bayesian Optimization, and CMA-ES. We also include results using the more recent, powerful, expensive, and text-only OpenAI o1-preview model on a subset of 10 examples from TraySim; these experiments used a maximum of 20 optimization iterations. The convergence trajectories show that o1-mini and o1-preview perform significantly better than GPT-4o in Phase 1 optimization. We see that LLMs initial convergence is fast, however with longer iterations CMA-ES appears to outperform in minimizing the trajectory error. However, Table 1 shows better results for LLMPHy. To gain insights into this discrepancy, in Figure 5(c), we plot the mean absolute error between the predicted physics parameters and their ground truth from the comparative methods. Interestingly, we see that LLMPHy estimations are better; perhaps because prior methods optimize variables without any semantics associated to them, while LLMPHy is optimizes “physics” variables, leading to the better performance and faster convergence. In Figure 5(b), we plot the convergence of LLMPHy Phase 2 iterations improving the PSNR between the synthesized (using the program) and the provide task images. As is clear, the correctness of the program improves over iterations. Both BO and CMA-ES are continuous methods and cannot optimize over the discrete space in Phase 2. However, LLMPHy is capable of optimizing in both continuous and discrete optimization spaces. We ought to emphasize this **important benefit**.

## 5 CONCLUSIONS AND LIMITATIONS

In this paper, we introduced the novel task of predicting the outcome of complex physical interactions, solving for which we presented LLMPHy, a novel setup combining an LLM with a physics engine. Our model systematically synergizes the capabilities of each underlying component, towards estimating the physics of the scene and experiments on our proposed TraySim dataset demonstrate LLMPHy’s superior performance. Notably, as we make no assumptions on the differentiability of the simulator, our framework could be considered as an LLM-based black-box optimization framework, leveraging LLMs’ knowledge for hyperparameter sampling. Our study shows that the recent powerful LLMs have enough world “knowledge” that combining this knowledge with a world model captured using a physics engine allows interactive and iterative problem solving for better reasoning.

While our problem setup is very general, we note that we only experiment with four physical attributes (albeit unique per each object class). While, this may not be limiting from a feasibility study of our general approach, a real-world setup may have other physics attributes as well that needs to be catered to. Further, we consider closed-source LVLMs due to their excellent program synthesis benefits. Our key intention is to show the usefulness of an LLM for solving our task and we hope future open-source LLMs would also demonstrate such beneficial capabilities.

<sup>4</sup>For LLMPHy, we are limited by the context window of the LLM and the cost.

## REFERENCES

- 540  
541  
542 Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Yoshua Bengio, Bernhard  
543 Schölkopf, Manuel Wüthrich, and Stefan Bauer. Causalworld: A robotic manipulation bench-  
544 mark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.
- 545  
546 S Aroca-Ouellette, C Paik, A Roncone, and K Kann. Prost: Physical reasoning of objects through  
547 space and time (arxiv: 2106.03634). arxiv, 2021.
- 548  
549 Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick.  
550 Phyre: A new benchmark for physical reasoning. *Advances in Neural Information Processing*  
551 *Systems*, 32, 2019.
- 552  
553 Fabien Baradel, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf. Cophy: Counter-  
554 factual learning of physical dynamics. *arXiv preprint arXiv:1909.12000*, 2019.
- 555  
556 Daniel M Bear, Elias Wang, Damian Mrowca, Felix J Binder, Hsiao-Yu Fish Tung, RT Pramod,  
557 Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, et al. Physion: Evaluating physical  
558 prediction from vision in humans and machines. *arXiv preprint arXiv:2106.08261*, 2021.
- 559  
560 Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical com-  
561 monsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*,  
562 volume 34, pp. 7432–7439, 2020.
- 563  
564 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared  
565 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large  
566 language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- 567  
568 Zhenfang Chen, Kexin Yi, Yunzhu Li, Mingyu Ding, Antonio Torralba, Joshua B Tenenbaum, and  
569 Chuang Gan. Comphy: Compositional physical reasoning of objects and events from videos.  
570 *arXiv preprint arXiv:2205.01089*, 2022.
- 571  
572 Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and  
573 trajectory planning algorithms: A general overview. *Motion and operation planning of robotic*  
574 *systems: Background and practical approaches*, pp. 3–27, 2015.
- 575  
576 Rohit Girdhar, Mannat Singh, Andrew Brown, Quentin Duval, Samaneh Azadi, Sai Saketh Ramb-  
577 hatla, Akbar Shah, Xi Yin, Devi Parikh, and Ishan Misra. Emu video: Factorizing text-to-video  
578 generation by explicit image conditioning. *arXiv preprint arXiv:2311.10709*, 2023.
- 579  
580 Amelia Glaese, Nat McAleese, Maja Trkebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Mari-  
581 beth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of  
582 dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*, 2022.
- 583  
584 Pei-Fu Guo, Ying-Hsuan Chen, Yun-Da Tsai, and Shou-De Lin. Towards optimizing with large  
585 language models. *arXiv preprint arXiv:2310.05204*, 2023.
- 586  
587 Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.
- 588  
589 Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution  
590 strategies. *Evolutionary computation*, 9(2):159–195, 2001.
- 591  
592 Hao Hao, Xiaoqun Zhang, and Aimin Zhou. Large language models as surrogate models in evolu-  
593 tionary algorithms: A preliminary study. *arXiv preprint arXiv:2406.10675*, 2024.
- 594  
595 Augustin Harter, Andrew Melnik, Gaurav Kumar, Dhruv Agarwal, Animesh Garg, and Helge Ritter.  
596 Solving physics puzzles by reasoning about paths. *arXiv preprint arXiv:2011.07357*, 2020.
- 597  
598 Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot  
599 planners: Extracting actionable knowledge for embodied agents. In *International conference on*  
600 *machine learning*, pp. 9118–9147. PMLR, 2022.

- 594 Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey  
595 Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM*  
596 *Joint European Software Engineering Conference and Symposium on the Foundations of Software*  
597 *Engineering*, pp. 1646–1656, 2023.
- 598  
599 Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair,  
600 Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source  
601 vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.
- 602 Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large  
603 language models are zero-shot reasoners. *Advances in neural information processing systems*,  
604 35:22199–22213, 2022.
- 605  
606 Mina Konakovic Lukovic, Yunsheng Tian, and Wojciech Matusik. Diversity-guided multi-objective  
607 bayesian optimization with batch evaluations. *Advances in Neural Information Processing Sys-*  
608 *tems*, 33:17708–17720, 2020.
- 609 Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ra-  
610 masesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative  
611 reasoning problems with language models. *Advances in Neural Information Processing Systems*,  
612 35:3843–3857, 2022.
- 613  
614 Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou,  
615 and Andrew M Dai. Mind’s eye: Grounded language model reasoning through simulation. *arXiv*  
616 *preprint arXiv:2210.05359*, 2022.
- 617 Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayara-  
618 man, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via  
619 coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
- 620  
621 Sean Memery, Mirella Lapata, and Kartic Subr. Simlm: Can language models infer parameters of  
622 physical systems? *arXiv preprint arXiv:2312.14215*, 2023.
- 623  
624 Jonas Mockus. On bayesian methods for seeking the extremum. In *Proceedings of the IFIP Techni-*  
625 *cal Conference*, pp. 400–404, 1974.
- 626  
627 Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. The importance of direc-  
628 tional feedback for llm-based optimizers. *arXiv preprint arXiv:2405.16434*, 2024.
- 629  
630 Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya  
631 Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*,  
632 2022.
- 633  
634 Ronan Riochet, Mario Ynocente Castro, Mathieu Bernard, Adam Lerer, Rob Fergus, Véronique  
635 Izard, and Emmanuel Dupoux. Intphys 2019: A benchmark for visual intuitive physics under-  
636 standing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5016–5025,  
637 2021.
- 638  
639 Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Matej Balog,  
640 M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang,  
641 Omar Fawzi, et al. Mathematical discoveries from program search with large language models.  
642 *Nature*, 625(7995):468–475, 2024.
- 643  
644 Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter  
645 Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using  
646 large language models. In *2023 IEEE International Conference on Robotics and Automation*  
647 *(ICRA)*, pp. 11523–11530. IEEE, 2023.
- 648  
649 Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su.  
650 LLM-planner: Few-shot grounded planning for embodied agents with large language models.  
651 In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009,  
652 2023.

- 648 Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a  
649 machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.  
650
- 651 Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world  
652 models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*,  
653 2024.
- 654 Kei Terayama, Masato Sumita, Ryo Tamura, and Koji Tsuda. Black-box optimization for automated  
655 discovery. *Accounts of Chemical Research*, 54(6):1334–1346, 2021.  
656
- 657 Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze  
658 Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog  
659 applications. *arXiv preprint arXiv:2201.08239*, 2022.
- 660 Yufei Tian, Abhilasha Ravichander, Lianhui Qin, Ronan Le Bras, Raja Marjeh, Nanyun Peng, Yejin  
661 Choi, Thomas L Griffiths, and Faeze Brahman. Macgyver: Are large language models creative  
662 problem solvers? *arXiv preprint arXiv:2311.09682*, 2023.  
663
- 664 Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control.  
665 In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033,  
666 2012. doi: 10.1109/IROS.2012.6386109.
- 667 Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry  
668 without human demonstrations. *Nature*, 625(7995):476–482, 2024.  
669
- 670 Yi Ru Wang, Jiafei Duan, Dieter Fox, and Siddhartha Srinivasa. Newton: Are large language models  
671 capable of physical reasoning? *arXiv preprint arXiv:2310.07018*, 2023.
- 672 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny  
673 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in  
674 neural information processing systems*, 35:24824–24837, 2022.
- 675 Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *Proceedings  
676 of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 699–707, 2017.  
677
- 678 Cheng Xue, Vimukthini Pinto, Chathura Gamage, Ekaterina Nikonova, Peng Zhang, and Jochen  
679 Renz. Phy-q: A benchmark for physical reasoning. *arXiv preprint arXiv:2108.13696*, 3, 2021.  
680
- 681 Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun  
682 Chen. Large language models as optimizers, 2024. URL <https://arxiv.org/abs/2309.03409>.  
683
- 684 Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial  
685 dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701



# Appendices

## TABLE OF CONTENTS

1. Qualitative Videos: See **LLMPhy\_comparison\_videos\_TraySim.mp4** in Suppl\_v2.zip.
2. Simulation Setup: A
3. TraySim Dataset: B
4. Physics Parameter Sensitivity: C
5. Details of LLMPhy Phases: D
  - (a) Phase 1 Prompt and Details: D.1
  - (b) Phase 2 Prompt and Details: D.2
6. Performances to Other LLMs: E
7. Ablation Studies: F
8. LLMPhy Detailed Convergence Analysis: G
9. Qualitative Results: H
10. LLMPhy Optimization Trace, Program Synthesis, and LLM Interactions: I
11. Example Synthesized Programs: J
12. LLMPhy Optimization and Interaction Trace (Phase 1): K
13. LLMPhy Optimization and Interaction Trace (Phase 2): L

## A SIMULATION SETUP

As discussed in the previous section, we are determining the physical characteristics of our simulation using a physics engine. MuJoCo Todorov et al. (2012) was used to setup the simulation and compute the rigid body interactions within the scene. It is important to note that any physics engine capable of computing the forward dynamics of a multi-body system can be integrated within our framework. This is because LLMPhy implicitly estimates the outcome of a scene based on the specific physical laws the engine is computing. To be clear, LLMPhy does not assume any physical model of the world and operates entirely as a black-box optimizer. The world model is entirely captured by the physics engine that executes the program LLMPhy produces.

The simulation environment is build upon a template of the *World*,  $\mathcal{W}$ , which contains the initial parametrization of our model of Newtonian physics. This includes the gravity vector  $\mathbf{g}$ , time step, and contact formulation, but also graphical and rendering parameters later invoked by the LLM when executing the synthesized program. MuJoCo uses internally a soft contact model to compute for instance complementarity constraints; in our implementation we use a non-linear sigmoid function that allows a very small inter-body penetration and increases the simulation stability during abrupt accelerations. We use elliptic friction cones to replicate natural contacts more closely. We further take advantage of the model architecture of MuJoCo by programmatically inserting arbitrary objects  $o_k$  from the classes in  $\mathcal{C}$  into the scene, as described in Section 3.1. For each parametric object class in  $\mathcal{C}$ , we generate an arbitrary appearance and physical attributes such as static friction, stiffness, damping, and armature. An arbitrary number of object instances are created from each class (up to a provided limit on their total number) and placed at randomly chosen positions on a regular grid (scene layout). The graphical renderer is used to record the frame sequences  $\mathbf{X}$  corresponding to five orthogonally placed cameras around the *World* origin, including a top-down camera. In addition, we support panoptic segmentation of all objects in the scene and store the corresponding masks for arbitrarily chosen key frames. The simulated data also contains privileged information such as the pusher-tray contact information (*i.e.* force, location, velocity, time stamp), and the stability information for each object,  $\mathcal{S}_k = \{1 | \arccos(\mathbf{g}, O\mathbf{z}_k) < \alpha, 0 | otherwise\}$ , where  $\mathbf{g}$  is the gravity vector,  $O\mathbf{z}_k$  is the upright direction of object  $k$  and  $\alpha$  is an arbitrarily chosen allowable tilt. Thus, in our experiments, we use  $\alpha = 45^\circ$ . Given that we consider only rigid objects with uniformly distributed mass, we assume that this a reasonable and conservative threshold.

Other than the physics parametrization of each object class  $\mathcal{C}$  and the scene layout  $\cup o_k$ , the outcome of the simulation for sequence  $\mathbf{X}$  is given by the initial conditions of the pusher object  $p$ , namely its initial velocity  $\dot{\mathbf{p}}_s$  and position  $\mathbf{p}_s$ . The usual torque representation is used:

$$\boldsymbol{\tau} = \mathbf{I}_C \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}_C \boldsymbol{\omega}, \quad (3)$$

which relates the angular acceleration  $\alpha$  and angular velocity  $\dot{\boldsymbol{\omega}}$  to the objects torque  $\boldsymbol{\tau}$ . The simulator computes in the end the motion of each object based on the contact dynamics model given by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{S}_a^T \boldsymbol{\tau} + \mathbf{S}_u^T \boldsymbol{\lambda}_u + \mathbf{J}_c^T(\mathbf{q}) \boldsymbol{\lambda}_c, \quad (4)$$

where  $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{(n_a+n_u) \times (n_a+n_u)}$  is the mass matrix;  $\mathbf{q} \triangleq [\mathbf{q}_a^T, \mathbf{q}_u^T]^T \in \mathbb{R}^{n_a+n_u}$  are generalized coordinates; and  $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n_a+n_u}$  represents the gravitational, centrifugal, and the Coriolis term. The selector matrices  $\mathbf{S}_a = [\mathbb{I}_{n_a \times n_a} \mathbf{0}_{n_a \times n_u}]$  and  $\mathbf{S}_u = [\mathbf{0}_{n_u \times n_a} \mathbb{I}_{n_u \times n_u}]$  select the vector of generalized joint forces  $\boldsymbol{\tau} \in \mathbb{R}^{n_a}$  for the *actuated* joints  $n_a$ , or  $\boldsymbol{\lambda}_u \in \mathbb{R}^{n_u}$  which are the generalized contact forces of the *unactuated* DOF created by the dynamics model, respectively.  $\mathbf{J}_c(\mathbf{q}) \in \mathbb{R}^{6n_c \times (n_a+n_u)}$  is the Jacobian matrix and  $\boldsymbol{\lambda}_c \in \mathbb{R}^{6n_c}$  are the generalized contact forces at  $n_c$  contact points. In our simulated environment, only the pusher object  $p$  has actuated joints which sets its initial velocity and heading, while the rest of the joints are either unactuated or created by contacts. The state of the system is represented by  $\mathbf{s} \triangleq [\mathbf{q}^T \dot{\mathbf{q}}^T]^T$ .

## B TRAYSIM DATASET

Using the simulation setup described in Sec A, we created 100 task sequences using object classes  $\mathcal{C} = \{\text{wine glass, martini glass, bottle}\}$  with object instances from these classes arranged roughly in a  $3 \times 3$  matrix on the tray. The instance classes and the number of instances are randomly chosen with a minimum of 5 and a maximum of 9. Each task sequence is associated with an auxiliary sequence for parameter estimation that contains at least one object instance from every class of object appearing in the task images. For example, if a task image (that is, the first image in a task sequence) has 3 bottles, then we will have a bottle in the auxiliary sequence. We assume each instance is defined by a triplet: (color, type, location), where the color is unique across all the instances on the tray so that it can be identified across the multi-view images, especially when some views occlude some of the instances. The physical parameters of the objects are assumed to be the same for both the task sequences and the auxiliary sequences, and instances of the same object classes have the same physical parameters. The physics parameters were randomly sampled for each problem in the dataset. We assume the pusher is placed at the same location in both auxiliary and task data; however this location could be arbitrary and different and will not affect our experiments as such locations will be supplied to the simulator in the respective phases and are not part of inference.

**Ground Truth Physics:** When generating each problem instance in the TraySim dataset, the physical parameters of the object classes are randomly chosen within the following ranges: sliding friction in  $(0.1, 1]$ , inertia and stiffness in  $(0, 1)$ , and damping in  $(0, 10)$ . We assume a fixed and known mass for each object type across problem instances, namely we assume a mass of 20 units for bottle, 10 units for martini glass, and 4 units for the wine glass. The tray used a mass of 0.5 and the pusher with a mass 20. Further, for both the task and the auxiliary sequences we assume the pusher is located at the same initial location in the scene. However, for all the auxiliary sequences, we assume the pusher moves with an initial  $(x, y)$  velocity of  $(-4.8, -4.8)$  m/s towards the tray, while for the task sequences, this velocity could be arbitrary (but given in the problem question), with each component of velocity in the range of  $[-7, -3]$  m/s. We further assume that the pusher impact direction coincides with the center of the circular tray in all problem instances.

**Optimization Space:** We note that each object class has a unique physics, i.e., each object class has its own friction, stiffness, damping, and inertia, which are different from other object classes. However, instances of the same class share the same physics. Thus, our optimization space for physics estimation when using 3 object instances, each one from a unique class, is thus 12. For the Phase 2 optimization, the LLM has to reason over the object classes for each object instance in the layout image, their positions in the  $3 \times 3$  grid, and their colors. This is a sufficiently larger optimization space, with 10 instance colors to choose from, 3 object classes, and 9 positions on the grid.

**Additional Objects:** In addition to the setup above that we use for the experiments in the main paper, we also experiment with additional object classes in this supplementary materials to show

810 the scalability of our approach to more number of parameters to optimize. To this end, we consider  
 811 two additional object classes, namely: i) *flute\_glass* with a mass of 15.0, and *champagne\_glass*, with  
 812 again a mass of 15.0. The physics parameters for these classes are sampled from the same range  
 813 described above. Even when we use these additional classes, the layout uses the same  $3 \times 3$  matrix  
 814 for phase 2, however their Phase 1 evaluation has now  $5 \times 4$  variables to optimize instead of 12. We  
 815 created 10 sequences with these additional objects, as our goal is to ablate on the scalability of our  
 816 approach, than running on a full evaluation as against the results reported in the main paper.

817 **Simulation and QA Task:** Each sequence was rendered using the simulator for 200 time steps,  
 818 each step has a duration of 0.01s. We used the last video frame from the task sequence to check  
 819 the stability of each instance. Specifically, if the major axis of an object instance in the last frame  
 820 of a task sequence makes an angle of more than 45 degrees with the ground plane, then we deem  
 821 that instance as *stable*. We randomly select five object instances and create a multiple choice  
 822 candidate answer set for the question-answering task, where the ground truth answer is the subset of  
 823 the candidates that are deemed upright in the last frame. Our QA question is “Which of the object  
 824 instances on the tray will remain upright when the tray is impacted by a pusher with a velocity of  $(x,$   
 825  $y)$  m/s from the location  $(loc_x, loc_y)$  in a direction coinciding with the center of the tray“. Without  
 826 any loss of generality, we assume  $(loc_x, loc_y)$  are fixed in all cases, although as it is a part of the  
 827 question and is simulated (and not inferred) any other location of the tray or the pusher will be an  
 828 issue when inferring using LLMPHy. From an evaluation perspective, keeping the pusher too close  
 829 to the tray may result in all object instances toppling down, while placing it far with smaller velocity  
 830 may result in the pusher halting before colliding with the tray. Our choice of the pusher velocity  
 831 was empirically selected such that in most cases the outcome of the impact is mixed and cannot be  
 832 guessed from the setup.

## 833 C PHYSICS PARAMETER SENSITIVITY

835 A natural question one may ask about the TraySim dataset is “*how sensitive are the physics param-*  
 836 *eters to influence the outcome?* In Figure 6, we show three Phase 1 sequences consisting of the same  
 837 objects and their layout, however varying the physics attributes as shown in the histogram plots. The  
 838 pusher velocity is fixed for all the sequences. As can be seen from the figure, varying the parameters  
 839 result in entirely different stability for the objects after the impact, substantiating that the correct  
 840 inference of these parameters is important to reproduce the correct the outcome.

## 842 D DETAILS OF LLMPHY PHASES

843 In this section, we detail the inputs and expected outputs provided in each phase of LLMPHy.  
 844

### 847 D.1 PHASE 1 PROMPT AND DETAILS

848 In this phase, we provide as input to the LLM four items: i) a prompt describing the problem  
 849 setup, the qualitative parameters of the objects (such as mass, height, size of tray, etc.) and the  
 850 task description, ii) an in-context example consisting of sample trajectories of the object instances  
 851 from its example auxiliary sequence, iii) a program example that, for the given example auxiliary  
 852 sequence trajectories, shows their physical parameters and the output structure, and iv) auxiliary task  
 853 sequence trajectories (from the sequence for which the physical parameters have to be estimated)  
 854 and a prompt describing what the LLM should do. The in-context example is meant to guide the  
 855 LLM to understand the setup, the program structure we expect the LLM to produce, and our specific  
 856 APIs that need to be called from the synthesized program. Figure 7 shows the prompt preamble we  
 857 use in Phase 1. Please see our Appendix I for the precise example of the full prompt that we use.  
 858 Figure 7 (bottom) shows an example trajectories LLM should optimizes against.

859 When iterating over the LLM predictions, we augment the above prompt with the history of all the  
 860 estimations of the physical parameters that the LLM produced in the previous iterations (extracted  
 861 from the then generated code) and the  $\ell_2$  norm between the generated and ground truth object tra-  
 862 jectories for each object instance in the auxiliary sequence, with an additional prompt to the LLM  
 863 as follows: “*We ran your code in our simulator using the physical parameters you provided below...  
 The error in the prediction of the trajectories using these physical parameters is given below. Can*

864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917

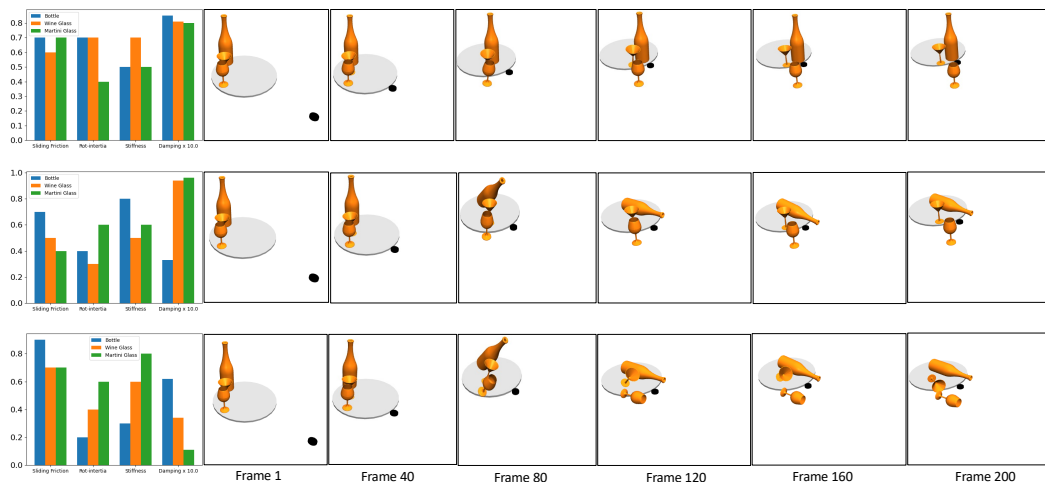


Figure 6: Illustration of the changes in the physical parameters (left histogram, sliding friction, rotation inertia, stiffness, and damping, respectively), and the result of the impact on three objects placed at the same location on the tray (Frame 1) and being impacted by the same force from the pusher. The examples are from the Phase 1 of our dataset. As is clear in the last frame (Frame 200) that changes in the the physical parameters results in entirely different outcomes, substantiating that the estimations of these parameters is important in solving our task.

*you refine your code to make the trajectories look more similar to the ones in given in ...? Your written code should strictly follow the same code structure as provided in ...*". See Figure 8 for an example. While, we may use computer vision methods for estimating the trajectory of motion of the objects in this Phase, i.e.,  $\tau$  function in (1), in this work we directly use the trajectories from the simulator for optimization for two reasons: i) we assume the Phase 1 allows complete access to the objects and the setup for parameter estimation, and ii) the focus of this phase is to estimate the physics parameters assuming everything else is known, while the perception task is dealt with in Phase 2. In a real-world setup, we may use AprilTags for producing the object trajectories. This simulation trajectory for Phase 1 will also be provided as part of our TraySim dataset, while also providing the multiview Phase 1 videos for anyone to use vision foundation models for solving the perception problem.

## D.2 PHASE 2 PROMPT AND DETAILS

The goal of the LLM in Phase 2 is to predict the object instance triplet from the multi-view task images. Towards this end, the LLM generates code that incorporates these triplets, so that when this code is executed, the simulator will reproduce the scene layout. Similar to Phase 1, we provide to the LLM an in-context example for guiding its code generation, where this in-context example contains multi-view images and the respective program, with the goal that the LLM learns the relation between parts of the code and the respective multi-view images, and use this knowledge to write code to synthesize the layout of the provided task images. When iterating over the optimization steps, we compute an error feedback to the LLM to improve its previously generated code, where the feedback consists of the following items: i) the program that the LLM synthesized in the previous optimization step, ii) the PSNR between the task image and the simulated image (top-down views), and iii) the color of the object instances in error<sup>5</sup>. Using this feedback, the Phase 2 LLM is prompted to fix the code associated with the triplets in error. Our feedback prompt in Phase 2 thus looks like in the following example: *"The chat history below shows a previous attempt of GPT-4o in generating Python code to reproduce the task images .... For each attempt, we ran the GPT-4o generated code in our simulator and found mistakes. Below we provide the code GPT produced, as well as the PSNR of the generated image against the given top-down image. Can you refine your code to*

<sup>5</sup>This is done by inputting a difference image (between the task and synthesized images) to another vision-and-language LLM that is prompted to identify the triplets that are in error

918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971

**Prompt Preamble:** The given scene has a tray with three objects (a bottle, a wine\_glass, and a martini\_glass) on it. The radius of the tray is 1.8 and its center of gravity is 0.05 above the ground with a sliding friction of 0.1 and no spin or roll friction. The radius of bottle is 0.4 and its center of gravity is 1.1 above the ground. The center of gravity of the martini\_glass is at a height of 0.5. The center of gravity of the wine\_glass is 0.9 above the ground. The tray is impacted by a pusher and the tray with the objects on it moves. Python code in example\_code\_1.py creates the scene and runs the simulation. The trajectories in object\_traj\_example\_1.txt show the motion of the center of gravity of the objects when running the simulation. Your task is to analyze the given example and then write similar code to produce the trajectories given in 'problem\_trajectories.txt'.

You must assume the scene is similar to the one given, however the physics between the tray and the objects are different, that is, the sliding-friction, damping, stiffness, and armature need to be adjusted for all the physical\_parameters\_for\_object\_id\_\* dictionaries in the example\_code\_1.py so as to reproduce the trajectories in 'problem\_trajectories.txt'. You must assume that the physics of the tray with the ground remains the same and so is the external force applied on the tray by the pusher. The trajectories use a timestep of 0.2s. Do not attempt to change the physics parameters beyond their first significant digit. Your written code should strictly follow the same code structure as provided in example\_code\_1.py. You may further assume that multiple instances of the same object will have the same physical parameters.

You must not change the 'mass' of the objects in your generated code. Do not include the object trajectories in your generated code as that will fail our simulator.

Note that the simulation trajectory in problem\_trajectories.txt may use instances of bottle, martini\_glass, and wine\_glass. The name of the objects is provided in the problem\_trajectories.txt file. The mass for the objects are as follows: wine\_glass is 4.0, martini\_glass is 10.0 and bottle is 20.0

```

# nexample_code_1.py
sim = SIMULATOR_MODEL()
sim.create_pusher(3.0 3.0 0.05)
physical_parameters_for_object_id_tray = {
    'sliding-friction': 0.1,
    'armature': 0.1,
    'stiffness': 0.0,
    'mass': 0.5,
    'damping': 20
}
sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
physical_parameters_for_object_id_1 = {
    'sliding-friction': 0.1,
    'armature': 0.2,
    'stiffness': 0.3,
    'mass': 20.0 # 'mass' is 20.0 for bottle, 10.0 for martini_glass, and 5.0 for wine_glass
    'damping': 5.7
}
sim.create_object(object_id=1, object_name='bottle', object_location=(row_1', 'column_3'), object_color='orange', object_physics=physical_parameters_for_object_id_1)
physical_parameters_for_object_id_2 = {
    'sliding-friction': 0.5,
    'armature': 0.4,
    'stiffness': 1.0,
    'mass': 10.0 # 'mass' is 20.0 for bottle, 10.0 for martini_glass, and 5.0 for wine_glass
    'damping': 8.8
}
sim.create_object(object_id=2, object_name='martini_glass', object_location=(row_1', 'column_2'), object_color='orange', object_physics=physical_parameters_for_object_id_2)
...
sim.create_scene()
sim_out=sim.run_simulation()
del sim

```

object\_traj\_example\_1.txt

```
tray_motion_trajectory (x, y, z) = [(0.0, 0.0, 0.1), (-0.8, -0.8, 0.1), (-1.4, -1.4, 0.1), (-1.8, -1.8, 0.1), (-2.1, -2.1, 0.1), (-2.3, -2.3, 0.1), (-2.4, -2.5, 0.1), (-2.6, -2.6, 0.1), (-2.7, -2.7, 0.1)]
```

```
bottle_motion_trajectory (x, y, z) = [(-1.1, -1.1, 1.1), (-1.1, -1.1, 1.1), (-1.1, -1.1, 1.1), (-1.1, -1.1, 1.1), (-1.2, -1.2, 1.1), (-1.3, -1.3, 1.1), (-1.4, -1.5, 1.1), (-1.5, -1.6, 1.1), (-1.6, -1.7, 1.1)]
```

...

```
wine_glass_motion_trajectory (x, y, z) = [(-1.0, 1.0, 0.9), (-1.1, 0.9, 1.0), (-1.1, 0.9, 0.8), (-1.2, 0.9, 0.8), (-1.2, 0.9, 0.8), (-1.3, 0.8, 0.8), (-1.3, 0.8, 0.8), (-1.2, 0.8, 0.8)]
```

problem\_trajectories.txt

```
tray_motion_trajectory (x, y, z) = [(0.0, 0.0, 0.1), (-0.7, -0.7, 0.1), (-1.1, -1.1, 0.1), (-1.4, -1.4, 0.1), (-1.6, -1.6, 0.1), (-1.8, -1.8, 0.1), (-2.0, -2.0, 0.1), (-2.1, -2.1, 0.1), (-2.2, -2.2, 0.1)]
```

```
bottle_motion_trajectory (x, y, z) = [(-1.1, -1.1, 1.1), (-1.1, -1.1, 1.1), (-1.3, -1.3, 1.1), (-1.4, -1.5, 1.1), (-1.5, -1.6, 1.0), (-1.5, -1.6, 0.9), (-1.5, -1.7, 0.6), (-1.5, -1.7, 0.5), (-1.6, -1.8, 0.5)]
```

...

```
wine_glass_motion_trajectory (x, y, z) = [(-1.0, 1.0, 0.9), (-1.1, 0.9, 1.0), (-1.2, 0.8, 0.9), (-1.4, 0.8, 0.8), (-1.6, 0.7, 0.8), (-1.6, 0.7, 0.8), (-1.6, 0.7, 0.8), (-1.6, 0.7, 0.8), (-1.5, 0.7, 0.8)]
```

Figure 7: Top: Prompt preamble used in LLMPhy Phase 1. Middle: The example program provided to the LLM to inform the structure of the to be synthesized program. Bottom: The object instance trajectories provided as input to the LLM. There are two types of trajectories: i) example trajectories (named “object\_traj\_example\_1.txt” for the program example, and ii) “problem\_trajectories.txt” which the LLM should produce the program for.



972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

```

We ran your code in our simulator using the physical parameters you provided below in physical_parameters_for_object_id_* dictionaries. The error in the prediction of the trajectories using these physical parameters is given below. Can you refine your code to make the trajectories look more similar to the ones in 'problem_trajectories.txt'? Your written code should strictly follow the same code structure as provided in example_code_1.py.

GPT Attempt #1 produced :

physical_parameters_for_object_id_1 = {
'sliding-friction': 0.2, # Adjusted
'armature': 0.3, # Adjusted
'stiffness': 0.4, # Adjusted
'mass': 20.0, # Mass remains unchanged
'damping': 6.0 # Adjusted
}

physical_parameters_for_object_id_2 = {
'sliding-friction': 0.3, # Adjusted
'armature': 0.4, # Adjusted
'stiffness': 0.7, # Adjusted
'mass': 10.0, # Mass remains unchanged
'damping': 7.0 # Adjusted
}

physical_parameters_for_object_id_3 = {
'sliding-friction': 0.3, # Adjusted
'armature': 0.5, # Adjusted
'stiffness': 0.4, # Adjusted
'mass': 4.0, # Mass remains unchanged
'damping': 6.0 # Adjusted
}

Trajectory Prediction Error using the above parameters is listed below:
Trajectory Error for bottle = 1.25
Trajectory Error for martini_glass = 2.17
Trajectory Error for wine_glass = 1.22
Total Trajectory Error (including tray)= 6.62
Average Trajectory Error (including tray)= 1.66

```

Figure 8: The prompt shows the LLM feedback, where the parameters from the physical parameter snippet from the synthesized program are extracted to produce the prompt along with the errors the executed code produced (against the trajectories in “problem\_trajectories.txt”) on each object class. We append all previous responses consecutively when sending the new query to the LLM.

*reproduce the task images correctly? You should not change any part of the code corresponding to correctly inferred objects. <code ...>. Colors of the objects in the code above that are misplaced: colors = { 'orange', 'purple', 'cyan' }. PSNR for the generated image against given top-down image = 39.2 Please check the locations of these objects in task\_image\_top\_view\_1.png and fix the code accordingly.”.* We show a full prompt for the Phase 2 LLM in Sec. I.

## E PERFORMANCES TO OTHER LLMs

In Table 3, we compare the performance of Phase 1 and Phase 2 of LLMPHy to various alternatives and prior black-box optimization methods. This table includes additional results than those reported in the main paper in Table 1. In Experiments 4–6, we compare to the various black-box optimization methods for estimating the physics parameters while keeping the Phase 2 inference from LLMPHy as in the Experiment 3. To be comparable, we used the same number of iterations for all the methods. As can be noted from the table, LLMPHy leads to better performances compared to other methods in reasoning on the impact outcomes. In Experiments 7–8, we also executed the prior methods for longer number of steps, which improved their performances, however they appear to be still below that of LLMPHy.

In Table 4, we compare the performances to other LLM choices in Phase 1 of LLMPHy. As the experiments that use OpenAI o1 model was conducted on a smaller subset of ten problems from the TraySim dataset, we report only the performance on this subset for all methods. We find that the o1 variant of the models demonstrate better performances against CMA-ES and substantially better than BO.

## F ABLATION STUDIES

In this section, we analyze various aspects of LLMPHy performance and is reported in Table 5. In addition to Avg. IoU performance as done in the main paper, we also report the ‘precise IoU’ that counts the number of times the predicted answer (i.e., the set of stable object instances listed in the answer options) match precisely with the ground truth.

1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038

Expt #	Phase 1	Phase 2	Avg. IoU (%)
1	Random	Random	19.0
2	N/A	LLM	32.1
3	Random	LLMPhy	50.8
4	BO (30 iterations)	LLMPhy	59.6
5	CMA-ES (30 iterations)	LLMPhy	59.7
6	LLMPhy (30 iterations)	LLMPhy	<b>62.0</b>
7	BO (100 iterations)	LLMPhy	61.0
8	CMA-ES (100 iterations)	LLMPhy	60.7
9	Ground Truth (GT)	LLMPhy	65.1
10	CMA-ES	GT	75.8
11	LLMPhy	GT	77.5

Table 3: Performance analysis of LLMPhy Phase 1 and Phase 2 combinations against various alternatives, including related prior methods. We report the intersection-over-union of the predicted answer options and the ground truth answers in the multiple choice solutions.

1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049

Expt #	Phase 1	Phase 2	Avg. IoU (%)
1	BO	LLMPhy	49.6
2	CMA-ES	LLMPhy	53.0
3	LLMPhy (GPT-4o)	LLMPhy	53.0
4	LLMPhy (o1-mini)	LLMPhy	55.3
5	LLMPhy (o1)	LLMPhy	<b>57.0</b>

Table 4: Performance analysis (on a small subset of 10 examples) of LLMPhy Phase 1 and Phase 2 combinations against various alternatives using various LLMs within LLMPhy.

1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079

1. *How will LLMPhy scale to more number of object classes?* To answer this question, we extended the TraySim dataset with additional data with five object classes  $\mathcal{C} = \{\text{bottle, martini\_glass, wine\_glass, flute\_glass, champagne\_glass}\}$ . The last two items having the same mass of 15.0. We created 10 examples with this setup for our ablation study and re-ran all methods on this dataset. Figure 9 show an example of this setup using 5 object classes. The ablation study we report below use this setup. In Expt 1-3 in Table 5, we compare the performance of LLMPhy to BO and CMA-ES. We see that LLM performs the best. We also repeated the experiment in Expt 4-6 using the ground truth (GT) Phase 2 layout, thus specifically evaluating on LLMPhy Phase 1 physics estimation. Again we see the clear benefit in using LLMPhy on both Avg. IoU and Precise IoU, underlining that using more objects and complicating the setup does not affect the performance of our model. We note that all the methods in tis comparison used the same settings, that is the number of optimization iterations was set to 30, and we used o1-mini for LLMPhy.

2. *Robustness of LLMPhy Performances?* A natural question is how well do LLMPhy perform in real world settings or when using a different simulation setup. While, it needs significant efforts to create a real-world setup for testing LLMPhy (e.g., that may need programming a robot controller for generating a precise impact for the pusher, etc.) or a significant work to create APIs for a different simulator, we may test the robustness of the framework artificially, for example, by injecting noise to the feedback provided to the LLM/VLM at each iteration. We attempted this route by adding a noise equal to 25% of the smallest prediction error for each of the object instance trajectories in Phase 1. Specifically, we compute  $\ell_2$  error between the predicted and the provided object trajectory for each object class in Phase 1 of LLMPhy (let’s call it  $\{e_k\}_{k=1}^5$ ), computed the minimum of these errors say  $e_m$ , and replaced as  $\hat{e}_k := e_k + e_m \cdot \zeta / 4.0$  for  $k = 1, 2, \dots, 5$  and  $\zeta \sim \mathcal{N}(0, 1)$ . This will make the LLM essentially uncertain about its physical parameter predictions, while the error (which is sufficiently high given the usual range of the error is between 0.5-4) simulates any underlying errors from a real physical system or simulation errors when using another physics engine. Our results in Expt. 7-8 in Table 5 show that LLMPhy is not very much impacted by the noise. While there is a drop of about 5% in accuracy (72.5% to 67.2%) when using GT, it is still higher than for example, when using CMA-ES on this additional dataset.

1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090

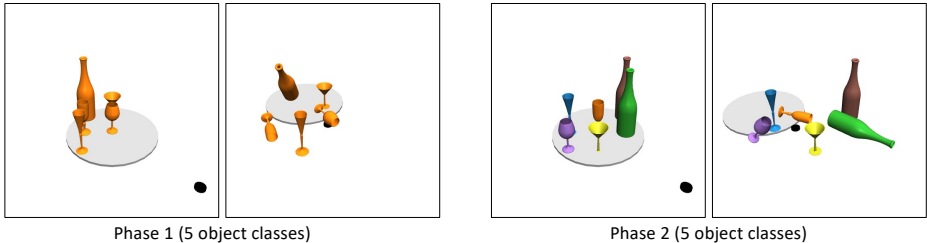


Figure 9: An example illustrating our extended dataset with 5 object classes.

1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099

3. *Advantage of using Optimization Trace?* As we alluded to early on in the paper, one of the differences from prior work such as Ma et al. (2023) is that LLMPHy uses the optimization trace against only the last feedback. In Table 5 Expt 9-10, we compare the performance when not using the full optimization trace. We see a drop of 5% (i.e., 56.4% Avg. IoU to 51.1%) showing that the optimization trace is useful. While using the optimization trace may demand longer context windows, we believe it also helps the LLM to avoid reconsidering previously generated parameter values and thus aids better convergence, especially for black-box optimization approaches, unless there is a provision to include a summary of the optimization trajectory to the LLM in another manner.

1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111

Expt #	Phase 1	Phase 2	Avg. IoU (%)	Precise IoU(%)
1	BO	LLMPHy	51.2	0.0
2	CMA-ES	LLMPHy	39.5	0.0
3	LLMPHy	LLMPHy	<b>56.4</b>	11.0
4	BO	GT	71.0	11.0
5	CMA-ES	GT	63.2	22.0
6	LLMPHy	GT	<b>72.5</b>	<b>33.0</b>
7	LLMPHy + noise	LLMPHy	52.1	22.0
8	LLMPHy + noise	GT	67.2	22.0
9	LLMPHy (last-only)	LLMPHy	51.1	11.0
10	LLMPHy (last-only)	GT	70.5	33.0

1112  
1113  
1114  
1115  
1116  
1117

Table 5: Performance comparison of LLMPHy against alternatives on various scene conditions and when using more number of objects on the simulated tray. In the experiments that show LLMPHy+noise, we perturb the object trajectories with 25% noise so that LLMPHy receives a noisy feedback. In the experiments LLMPHy (last-only), we feedback to LLMPHy only error and the physics parameters from the last iteration, without the full optimization trace.

1118  
1119  
1120

## G LLMPHY DETAILED CONVERGENCE ANALYSIS

1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129

In Figure 10(a), we plot the mean convergence (over a subset of the dataset) when using o1-preview, GPT-4o, o1-mini, Bayesian Optimization, and CMA-ES. We see that the o1 model, that is explicitly trained for solving scientific reasoning, appears to be beneficial in our task. Interestingly, we see that o1’s initial convergence is fast, however with longer iterations CMA-ES appears to outperform in minimizing the trajectory error. That being said, the plots in Figure 5(c) and Table 1 points out that having lower trajectory error does not necessarily imply the physical parameters are estimated correctly (as they are implicitly found and are non-linear with regards to the trajectories), and having knowledge of physics in optimization leads to superior results.

1130  
1131  
1132  
1133

Further to this, in Figure 10(d), we plot the histogram of best Phase 1 iterations between the various algorithms. Recall that the optimization methods we use are not based on gradients, instead are sampled discrete points, and the optimization approach is to select the next best sample towards minimizing the error. The plot shows that LLMPHy results in its best sample selections happen early on in its iterations than other methods.

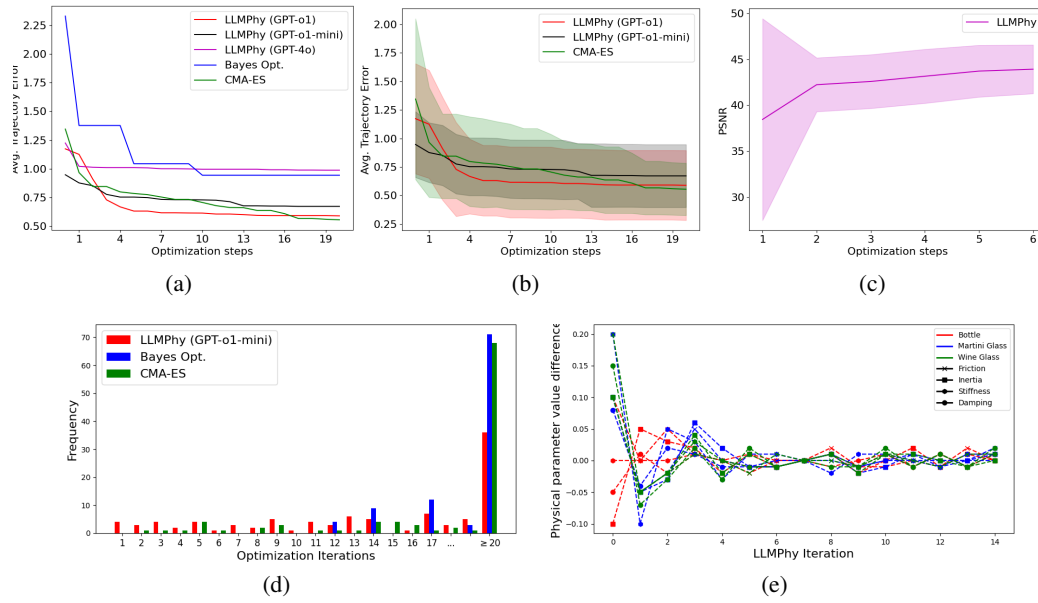


Figure 10: (a) shows comparison of convergence when using various state-of-the-art LLMs in LLMPhy against Bayesian optimization and CMA-ES. We plot the minimum loss computed thus far in the optimization process against the number of optimization steps. (b) plots show the convergence of LLMPhy and the error variance for Phase 1. (c) plots the convergence in Phase 2. We also compare the convergence using OpenAI o1-preview model as the LLM used in LLMPhy. (d) Histogram of the best optimization iteration when using LLMPhy against other methods. (e) shows the differences between subsequent values for the various physical parameters in a typical iteration of LLMPhy from its value in the previous iteration.

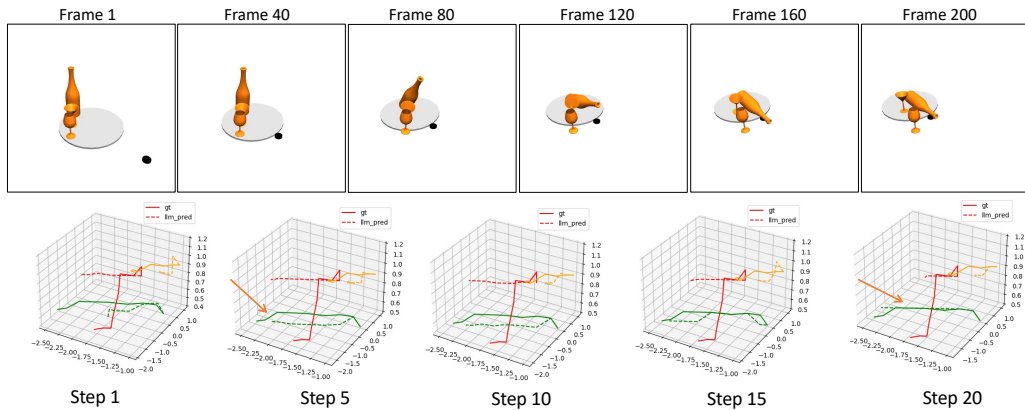


Figure 11: We show an example Phase 1 sequence (top). Below, we plot the motion trajectories for each of the objects in the frames and the predicted trajectories by LLMPhy from the optimization steps. The trajectory plots (below) show the ground truth trajectory (gt) and the predicted trajectory (llm\_pred), and as the iterations continue, we can see improvements in the alignment of the predicted and the ground truth object trajectories (as pointed out by the arrows).

In Figure 10(e), we plot the optimization parameter trace for one sample sequence, where we plot the differences between the values of the physics parameters produced by the LLM at an iteration against the values from the previous iteration. The plot shows the relative magnitude of changes the LLM makes to the parameters towards adjusting for the object trajectory error. We plot these adjustments for all the three objects and all the four parameters together in one plot so as to see the

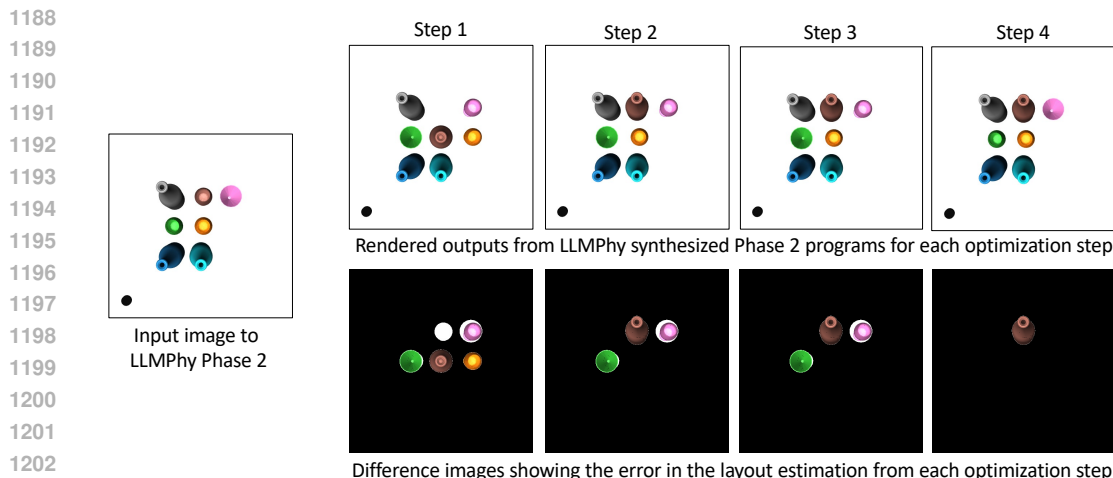


Figure 12: We show qualitative results from LLMPhy Phase 2 iterations. The input Phase 2 image is shown on the left. The top row shows the images produced by the simulator using the layout prediction code generated by LLMPhy for each Phase 2 optimization step. Below, we show the difference image between the predicted and the input Phase 2 images, clearly showing the errors. In Phase 2, the feedback to LLMPhy is produced using PSNR computed on the predicted and the ground truth images, as well as asking LLM (using the difference image) which of the objects are in error, and asking the LLM to fix the layout of these objects in the next iteration. As can be seen, the errors in the LLM layout prediction improves over iterations.

overall trend that the LLM makes. We also see that the LLM makes large adjustments in the first few iterations and it reduces in magnitude for subsequently. For this particular example, the LLMPhy converged in 15 iterations.

In Figure 5(a), we plot the convergence of LLMPhy-Phase 1, alongside plotting the variance in the trajectory error from the estimated physical parameters when used in the simulations. We found that a powerful LLM such as OpenAI o1-mini LLM or o1-preview demonstrates compelling convergence, with the lower bound of variance below that of other models. Our experiments suggest that better LLMs may lead to even stronger results.

In Figure 5(b), we plot the convergence of LLMPhy Phase 2 iterations improving the PSNR between the synthesized (using the program) and the provide task images. As is clear, their correctness of the program improves over iterations. We would like to emphasize that BO and CMA-ES are continuous optimization methods and thus cannot optimize over the discrete space of Phase 2 layout. This is an important benefit of using LLMPhy for optimization that can operate on both continuous and discrete state spaces.

## H QUALITATIVE RESULTS

In Figure 13, we show several qualitative results from our TraySim dataset and comparisons of LLMPhy predictions to those of BO and CMA-ES. In general, we find that when the velocity of the pusher is lower, and the sliding friction is high, objects tend to stay stable if they are heavier (e.g., a bottle), albeit other physics parameters also playing into the outcome. In Figure 11, we show example iterations from Phase 1 that explicitly shows how the adjustment of the physical parameters by LLMPhy is causing the predicted object trajectories to align with the ground truth. In Figure 12, we show qualitative outputs from the optimization steps in Phase 2, demonstrating how the error feedback to the LLM corrects its previous mistakes to improve the layout estimation.



1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295

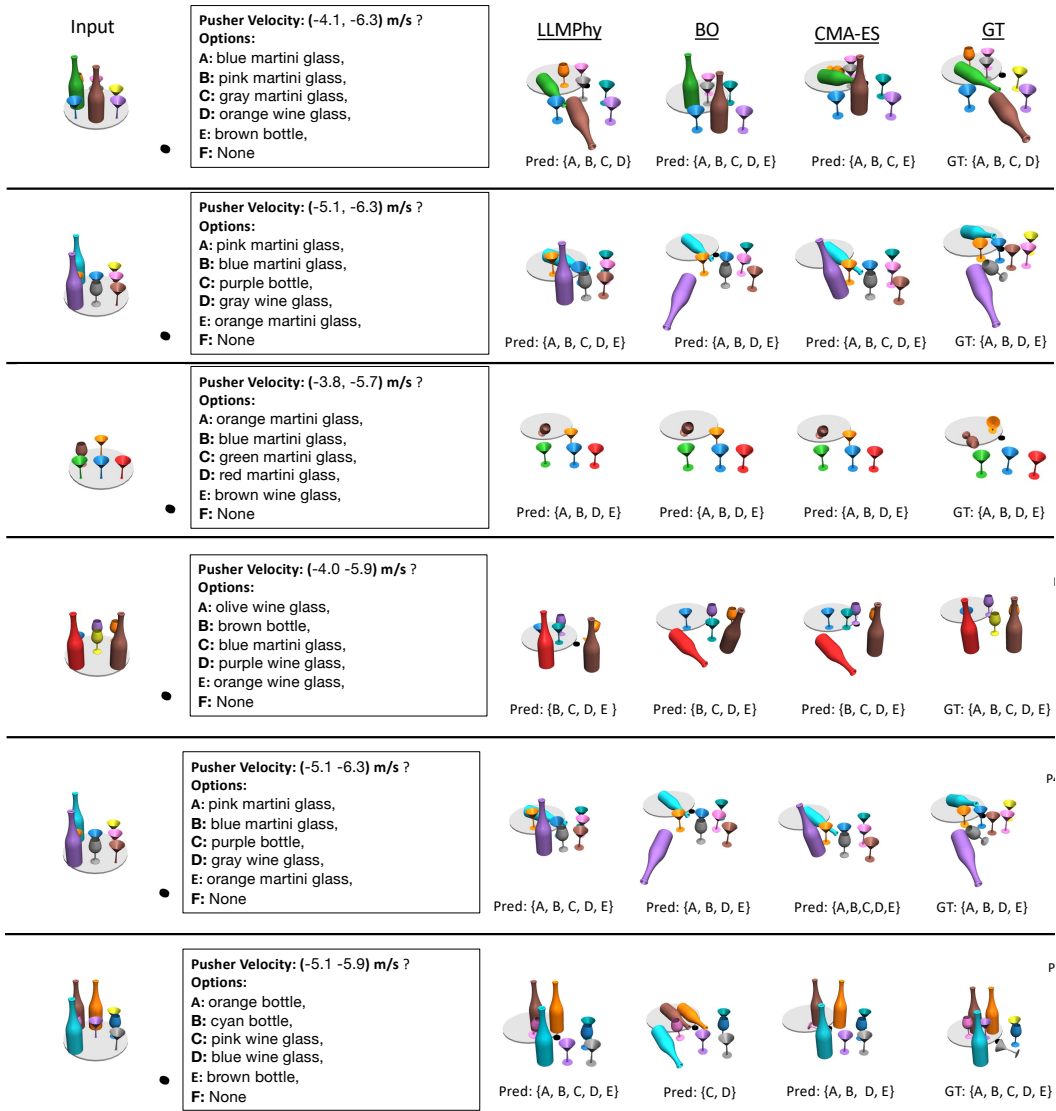


Figure 13: Qualitative comparisons between LLMPhy, Bayesian optimization, and CMA-ES.

## I LLMPHY OPTIMIZATION TRACE, PROGRAM SYNTHESIS, AND LLM INTERACTIONS

Below, we present the exact prompts we used for the LLM in our experiments for Phases 1 and 2, as well as depicting the programs LLM generate.

### Phase 1 Prompt:

The given scene has a tray with three objects (a bottle, a wine\_glass, and a martini\_glass) on it. The radius of the tray is 1.8 and its center of gravity is 0.05 above the ground with a sliding friction of 0.1 and no spin or roll friction. The radius of bottle is 0.4 and its center of gravity is 1.1 above the ground. The center of gravity of the martini\_glass is at a height of 0.5. The center of gravity of the wine\_glass is 0.9 above the ground. The tray is impacted by a pusher and the tray with the objects on it moves. Python code in example\_code\_1.py creates the scene and runs the simulation. The trajectories in object\_traj\_example\_1.txt show the motion of the center of gravity of the objects when running the simulation. Your task is to analyze the given example and then write similar code to produce the trajectories given in 'problem\_trajectories.txt'.

You must assume the scene is similar to the one given, however the physics between the tray and the objects are different, that is, the sliding-friction, damping, stiffness, and armature need to be adjusted for all the physical\_parameters\_for\_object\_id\_\* dictionaries in the example\_code\_1.py so as to reproduce the trajectories in 'problem\_trajectories.txt'. You must assume that the physics of the tray with the ground remains the same and so is the external force applied on the tray by the pusher. The trajectories use a time step of 0.2s. Do not attempt to change the physics parameters beyond their first significant digit. Your written code should strictly follow the same code structure as provided in example\_code\_1.py. You may further assume that multiple instances of the same object will have the same physical parameters.

You must not change the 'mass' of the objects in your generated code. Do not include the object trajectories in your generated code as that will fail our simulator.

Note that the simulation trajectory in problem\_trajectories.txt may use instances of bottle, martini\_glass, and wine\_glass. The name of the objects is provided in the problem\_trajectories.txt file. The mass for the objects are as follows: wine\_glass is 4.0, martini\_glass is 10.0 and bottle is 20.0.'

```
\# nexample\_code\_1.py
sim = SIMULATOR_MODEL()
sim.create_pusher('3.0 3.0 0.05')
physical_parameters_for_object_id_tray = {
    'sliding-friction': 0.1,
    'armature': 0.1,
    'stiffness': 0.0,
    'mass': 0.5,
    'damping': 20
}
sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
physical_parameters_for_object_id_1 = {
    'sliding-friction': 0.1,
    'armature': 0.2,
```

```

1350         'stiffness': 0.3,
1351         'mass': 20.0,
1352         'damping': 5.7
1353     }
1354     sim.create_object(object_id=1, object_name='bottle',
1355 object_location=('row_1', 'column_3'), object_color='orange',
1356 object_physics=physical_parameters_for_object_id_1)
1357     ...
1358
1359     sim.create_scene()
1360     sim_out=sim.run_simulation()
1361     del sim
1362
1363     # object\_traj\_example\_1.txt
1364     ...
1365
1366     bottle_motion_trajectory (x, y, z) = [(-1.1, -1.1, 1.1), (-1.1, -1.1,
1367 1.1), (-1.1, -1.1, 1.1), (-1.1, -1.1, 1.1), (-1.2, -1.2, 1.1), (-1.3,
1368 -1.3, 1.1), (-1.4, -1.5, 1.1), (-1.5, -1.6, 1.1), (-1.6, -1.7, 1.1)]
1369
1370     martini_glass_motion_trajectory (x, y, z) = [(-1.0, 0.0, 0.5), (-1.1,
1371 -0.0, 0.6), (-1.2, -0.1, 0.6), (-1.4, -0.4, 0.5), (-1.6, -0.6, 0.5),
1372 (-1.8, -0.8, 0.5), (-2.0, -0.9, 0.5), (-2.1, -1.0, 0.5), (-2.2, -1.1,
1373 0.5)]
1374     ...
1375
1376 Phase 2 Prompt:
1377 Attached are two images: 'example_1_top_down_view_1.png' (top-down view)
1378 and 'example_1_side_view_2.png' (side view) of the same scene. The top-
1379 down view shows a scene arranged roughly on a 3x3 grid. The scene was
1380 rendered using the code in 'example_code_1.py'. Objects in the scene
1381 belong to one of the following classes: {martini_glass, wine_glass,
1382 bottle} and can be one of the following colors: {purple, red, green,
1383 blue, olive, cyan, brown, pink, orange, gray}. Each color appears only
1384 once in the scene. Can you interpret the provided code using the images?
1385 Use the top-down image to determine the arrangement and color of the
1386 objects, and correlate this with the side view to identify the object
1387 classes. Each object instance has a unique color, helping you identify
1388 the same object across different views.
1389
1390 example_1_top_down_view_1.png
1391 Image: top-down-image url
1392 example_1_side_view_2.png
1393 Image: side-view image url
1394
1395 example_code_1.py
1396
1397 sim = SIMULATOR_MODEL()
1398 sim.create_pusher('3.0 3.0 0.05')
1399 physical_parameters_for_object_id_tray = {
1400     'sliding-friction': 0.1,
1401     'armature': 0.1,
1402     'stiffness': 0.0,
1403     'mass': 0.5,
1404     'damping': 20
1405 }
1406 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
1407 physical_parameters_for_object_id_1 = {
1408     'sliding-friction': 0.1,
1409     'armature': 0.2,

```

```
1404         'stiffness': 0.3,
1405         'mass': 20.0, # 'mass' is 20.0 for bottle, 10.0 for
1406             martini_glass, and 5.0 for wine_glass
1407         'damping': 5.7
1408     }
1409     sim.create_object(object_id=1, object_name='bottle', object_location=('
1410         row_2', 'column_3'), object_color='brown', object_physics=
1411         physical_parameters_for_object_id_1)
1412     physical_parameters_for_object_id_2 = {
1413         'sliding-friction': 0.6,
1414         'armature': 0.8,
1415         'stiffness': 0.6,
1416         'mass': 4.0, # 'mass' is 20.0 for bottle, 10.0 for
1417             martini_glass, and 5.0 for wine_glass
1418         'damping': 8.3
1419     }
1419     sim.create_object(object_id=2, object_name='wine_glass', object_location
1420         =('row_3', 'column_2'), object_color='pink', object_physics=
1421         physical_parameters_for_object_id_2)
1422     physical_parameters_for_object_id_3 = {
1423         'sliding-friction': 0.1,
1424         'armature': 0.2,
1425         'stiffness': 0.3,
1426         'mass': 20.0, # 'mass' is 20.0 for bottle, 10.0 for
1427             martini_glass, and 5.0 for wine_glass
1428         'damping': 5.7
1429     }
1429     sim.create_object(object_id=3, object_name='bottle', object_location=('
1430         row_1', 'column_1'), object_color='purple', object_physics=
1431         physical_parameters_for_object_id_3)
1432     physical_parameters_for_object_id_4 = {
1433         'sliding-friction': 0.1,
1434         'armature': 0.2,
1435         'stiffness': 0.3,
1436         'mass': 20.0, # 'mass' is 20.0 for bottle, 10.0 for
1437             martini_glass, and 5.0 for wine_glass
1438         'damping': 5.7
1439     }
1439     sim.create_object(object_id=4, object_name='bottle', object_location=('
1440         row_1', 'column_2'), object_color='olive', object_physics=
1441         physical_parameters_for_object_id_4)
1442     physical_parameters_for_object_id_5 = {
1443         'sliding-friction': 0.1,
1444         'armature': 0.2,
1445         'stiffness': 0.3,
1446         'mass': 20.0, # 'mass' is 20.0 for bottle, 10.0 for
1447             martini_glass, and 5.0 for wine_glass
1448         'damping': 5.7
1449     }
1449     sim.create_object(object_id=5, object_name='bottle', object_location=('
1450         row_3', 'column_1'), object_color='orange', object_physics=
1451         physical_parameters_for_object_id_5)
1452     physical_parameters_for_object_id_6 = {
1453         'sliding-friction': 0.5,
1454         'armature': 0.4,
1455         'stiffness': 1.0,
1456         'mass': 10.0, # 'mass' is 20.0 for bottle, 10.0 for
1457             martini_glass, and 5.0 for wine_glass
1458         'damping': 8.8
1459     }
```

```

1458 sim.create_object(object_id=6, object_name='martini_glass',
1459                   object_location=('row_2', 'column_2'), object_color='cyan',
1460                   object_physics=physical_parameters_for_object_id_6)
1461
1462 physical_parameters_for_object_id_7 = {
1463     'sliding-friction': 0.5,
1464     'armature': 0.4,
1465     'stiffness': 1.0,
1466     'mass': 10.0, # 'mass' is 20.0 for bottle, 10.0 for
1467                 martini_glass, and 5.0 for wine_glass
1468     'damping': 8.8
1469 }
1470 sim.create_object(object_id=7, object_name='martini_glass',
1471                   object_location=('row_2', 'column_1'), object_color='gray',
1472                   object_physics=physical_parameters_for_object_id_7)
1473
1474 physical_parameters_for_object_id_8 = {
1475     'sliding-friction': 0.5,
1476     'armature': 0.4,
1477     'stiffness': 1.0,
1478     'mass': 10.0, # 'mass' is 20.0 for bottle, 10.0 for
1479                 martini_glass, and 5.0 for wine_glass
1480     'damping': 8.8
1481 }
1482 sim.create_object(object_id=8, object_name='martini_glass',
1483                   object_location=('row_3', 'column_3'), object_color='green',
1484                   object_physics=physical_parameters_for_object_id_8)
1485
1486 physical_parameters_for_object_id_9 = {
1487     'sliding-friction': 0.1,
1488     'armature': 0.2,
1489     'stiffness': 0.3,
1490     'mass': 20.0, # 'mass' is 20.0 for bottle, 10.0 for
1491                 martini_glass, and 5.0 for wine_glass
1492     'damping': 5.7
1493 }
1494 sim.create_object(object_id=9, object_name='bottle', object_location=('
1495 row_1', 'column_3'), object_color='blue', object_physics=
1496 physical_parameters_for_object_id_9)
1497
1498 sim.create_scene()
1499 sim_out=sim.run_simulation()
1500 del sim
1501
1502 Using the above information, can you write code similar to '
1503 example_code_1.py' to reproduce the two images given below for a
1504 different scene? The images are named: 'task_image_top_down_view_1.
1505 png' for the top-down view of the scene and 'task_image_side_view_2.
1506 png' for the side-view of the same scene. Note that not all positions
1507 on the grid need to have an object.
1508
1509 task_image_top_view_1.png
1510 Image: top-down image url
1511 task_image_side_view_2.png
1512 Image: side-view image url
1513
1514 You should further use the following set of physical attributes for the
1515 respective objects in the scene when generating the code. Note that all
1516 the instances of the same object use the same physical attributes.
1517 object_name: bottle, mass: 20.0, 'sliding-friction': 0.3, 'armature':
1518 0.5, 'stiffness': 0.4, 'damping': 6.3
1519
1520 object_name: martini_glass, mass: 10.0, 'sliding-friction': 0.5, '
1521 armature': 0.3, 'stiffness': 1.2, 'damping': 8.3
1522
1523

```

```
1512 object_name: wine_glass, mass: 4.0, 'sliding-friction': 0.6, 'armature':
1513     0.7, 'stiffness': 0.8, 'damping': 7.8
1514
1515 The chat history below shows various attempts of GPT-4o in generating
1516 Python code to reproduce the task images (task_image_top_view_1.png,
1517 task_image_side_view_2.png). For each attempt, we ran the GPT-4o
1518 generated code in our simulator and found mistakes. Below we provide the
1519 code GPT produced in each attempt, titled GPT Attempt #K, where K is the
1520 attempt number, as well as the PSNR of the generated image against the
1521 given top-down image. Can you refine your code to reproduce the task
1522 images correctly? You should not change any part of the code
1523 corresponding to correctly inferred objects.
1524
1525 GPT Attempt #0:
1526 GPT generated code:
1527
1528 sim = SIMULATOR_MODEL()
1529 sim.create_pusher('3.0 3.0 0.05')
1530
1531 physical_parameters_for_object_id_tray = {
1532     'sliding-friction': 0.1,
1533     'armature': 0.1,
1534     'stiffness': 0.0,
1535     'mass': 0.5,
1536     'damping': 20
1537 }
1538 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
1539
1540 # Defining the physical parameters for each class of objects
1541 physical_parameters_for_bottle = {
1542     'sliding-friction': 0.3,
1543     'armature': 0.5,
1544     'stiffness': 0.4,
1545     'mass': 20.0,
1546     'damping': 6.3
1547 }
1548
1549 physical_parameters_for_wine_glass = {
1550     'sliding-friction': 0.6,
1551     'armature': 0.7,
1552     'stiffness': 0.8,
1553     'mass': 4.0,
1554     'damping': 7.8
1555 }
1556
1557 physical_parameters_for_martini_glass = {
1558     'sliding-friction': 0.5,
1559     'armature': 0.3,
1560     'stiffness': 1.2,
1561     'mass': 10.0,
1562     'damping': 8.3
1563 }
1564
1565 # Creating objects based on analysis of the task image
1566 sim.create_object(object_id=1, object_name='bottle', object_location=('
1567     row_2', 'column_1'), object_color='gray', object_physics=
1568     physical_parameters_for_bottle)
1569 sim.create_object(object_id=2, object_name='wine_glass', object_location
1570     =('row_1', 'column_1'), object_color='yellow', object_physics=
1571     physical_parameters_for_wine_glass)
1572 sim.create_object(object_id=3, object_name='wine_glass', object_location
1573     =('row_3', 'column_1'), object_color='blue', object_physics=
1574     physical_parameters_for_wine_glass)
```



```

1566 sim.create_object(object_id=4, object_name='martini_glass',
1567                  object_location=('row_2', 'column_2'), object_color='orange',
1568                  object_physics=physical_parameters_for_martini_glass)
1569 sim.create_object(object_id=5, object_name='bottle', object_location=('
1570 row_2', 'column_3'), object_color='red', object_physics=
1571 physical_parameters_for_bottle)
1572 sim.create_object(object_id=6, object_name='bottle', object_location=('
1573 row_3', 'column_2'), object_color='pink', object_physics=
1574 physical_parameters_for_bottle)
1575
1576 # Adding the missing martini glass with cyan color at row 1, column 2
1577 sim.create_object(object_id=7, object_name='martini_glass',
1578                  object_location=('row_1', 'column_3'), object_color='cyan',
1579                  object_physics=physical_parameters_for_martini_glass)
1580
1581 # Run the simulation
1582 sim.create_scene()
1583 sim_out = sim.run_simulation()
1584 del sim
1585
1586 Colors of the objects in the code above that are misplaced: {'purple', '
1587 cyan', 'pink'}
1588
1589 PSNR for the generated image against given top-down image = 40.9
1590 Please check the locations of these objects in task_image_top_view_1.png
1591 and fix the code accordingly.
1592
1593 End of Prompt

```

## J EXAMPLE OF SYNTHESIZED PROGRAMS

Below, we show an example of Python programs synthesized by LLMPhy, which are then executed to invoke the Mujoco Simulator to run and render the dynamical sequence, to extract the outcome of the simulation.

### Phase 1 program:

```

1599 sim = SIMULATOR_MODEL()
1600 sim.create_pusher('3.0 3.0 0.05')
1601 physical_parameters_for_object_id_tray = {
1602     'sliding-friction': 0.1,
1603     'armature': 0.1,
1604     'stiffness': 0.0,
1605     'mass': 0.5,
1606     'damping': 20
1607 }
1608 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
1609
1610 physical_parameters_for_object_id_1 = {
1611     'sliding-friction': 0.18, # finely adjusted as previous good attempt
1612     'armature': 0.1, # keeping consistent based on the effective value
1613     'stiffness': 0.16, # slight readjustment from previous effective value
1614     'mass': 20.0, # mass remains 20.0 for bottle
1615     'damping': 4.5 # slight decrease for finer adjustment
1616 }
1617
1618 physical_parameters_for_object_id_2 = {
1619     'sliding-friction': 0.35, # minor adjustment from better attempt
1620     'armature': 0.26, # further decreasing for better control
1621     'stiffness': 0.75, # slight decrease for better compliance

```

```

1620         'mass': 10.0,                # mass remains 10.0 for martini_glass
1621         'damping': 6.25             # slight adjustment for better trajectory control
1622     }
1623
1624     physical_parameters_for_object_id_3 = {
1625         'sliding-friction': 0.42,    # slight decrease for better fit
1626         'armature': 0.58,           # minor decrease for better control
1627         'stiffness': 0.39,          # minor decrease for better adjustment
1628         'mass': 4.0,                # mass remains 4.0 for wine_glass
1629         'damping': 6.55             # slight decrease for finer adjustment
1630     }
1631     sim.create_object(object_id=1, object_name='bottle',
1632                      object_location=('row_1', 'column_3'),
1633                      object_color='orange',
1634                      object_physics=physical_parameters_for_object_id_1)
1635
1636     sim.create_object(object_id=2, object_name='martini_glass',
1637                      object_location=('row_1', 'column_2'),
1638                      object_color='orange',
1639                      object_physics=physical_parameters_for_object_id_2)
1640
1641     sim.create_object(object_id=3, object_name='wine_glass',
1642                      object_location=('row_1', 'column_1'),
1643                      object_color='orange',
1644                      object_physics=physical_parameters_for_object_id_3)
1645
1646     sim.create_scene()
1647     sim_out = sim.run_simulation()
1648     del sim
1649
1650     Phase 2 program:
1651
1652     sim = SIMULATOR_MODEL()
1653     sim.create_pusher('3.0 3.0 0.05')
1654
1655     # Set physical attributes for the tray
1656     physical_parameters_for_object_id_tray = {
1657         'sliding-friction': 0.1,
1658         'armature': 0.1,
1659         'stiffness': 0.0,
1660         'mass': 0.5,
1661         'damping': 20
1662     }
1663     sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
1664
1665     # Physical parameters for each class of objects
1666     physical_parameters_bottle = {
1667         'sliding-friction': 0.9,
1668         'armature': 0.2,
1669         'stiffness': 0.3,
1670         'mass': 20.0,
1671         'damping': 6.2
1672     }
1673
1674     physical_parameters_wine_glass = {
1675         'sliding-friction': 0.7,
1676         'armature': 0.4,
1677         'stiffness': 0.6,
1678         'mass': 4.0,

```

```
1674     'damping': 3.4
1675 }
1676
1677 physical_parameters_martini_glass = {
1678     'sliding-friction': 0.7,
1679     'armature': 0.6,
1680     'stiffness': 0.8,
1681     'mass': 10.0,
1682     'damping': 1.1
1683 }
1684
1685 # Create objects with corrected positions and colors
1686 sim.create_object(object_id=1, object_name='wine_glass',
1687                  object_location=('row_1', 'column_1'),
1688                  object_color='gray',
1689                  object_physics=physical_parameters_wine_glass)
1690
1691 sim.create_object(object_id=2, object_name='wine_glass',
1692                  object_location=('row_1', 'column_2'),
1693                  object_color='red',
1694                  object_physics=physical_parameters_wine_glass)
1695
1696 sim.create_object(object_id=3, object_name='wine_glass',
1697                  object_location=('row_1', 'column_3'),
1698                  object_color='pink',
1699                  object_physics=physical_parameters_wine_glass)
1700
1701 sim.create_object(object_id=4, object_name='martini_glass',
1702                  object_location=('row_2', 'column_1'),
1703                  object_color='cyan',
1704                  object_physics=physical_parameters_martini_glass)
1705
1706 sim.create_object(object_id=5, object_name='bottle',
1707                  object_location=('row_2', 'column_2'),
1708                  object_color='yellow',
1709                  object_physics=physical_parameters_bottle)
1710
1711 sim.create_object(object_id=6, object_name='bottle',
1712                  object_location=('row_2', 'column_3'),
1713                  object_color='purple',
1714                  object_physics=physical_parameters_bottle)
1715
1716 sim.create_object(object_id=7, object_name='martini_glass',
1717                  object_location=('row_3', 'column_1'), object_color='blue',
1718                  object_physics=physical_parameters_martini_glass)
1719
1720 sim.create_object(object_id=8, object_name='bottle',
1721                  object_location=('row_3', 'column_2'),
1722                  object_color='orange',
1723                  object_physics=physical_parameters_bottle)
1724
1725 sim.create_object(object_id=9, object_name='martini_glass',
1726                  object_location=('row_3', 'column_3'),
1727                  object_color='brown',
1728                  object_physics=physical_parameters_martini_glass)
1729
1730 sim.create_scene()
1731 sim_out = sim.run_simulation()
1732 del sim
```

1728  
 1729  
 1730  
 1731  
 1732  
 1733  
 1734  
 1735  
 1736  
 1737  
 1738  
 1739  
 1740  
 1741  
 1742  
 1743  
 1744  
 1745  
 1746  
 1747  
 1748  
 1749  
 1750  
 1751  
 1752  
 1753  
 1754  
 1755  
 1756  
 1757  
 1758  
 1759  
 1760  
 1761  
 1762  
 1763  
 1764  
 1765  
 1766  
 1767  
 1768  
 1769  
 1770  
 1771  
 1772  
 1773  
 1774  
 1775  
 1776  
 1777  
 1778  
 1779  
 1780  
 1781

## K LLMPHY OPTIMIZATION AND INTERACTION TRACE (PHASE1)

Below, we provide the full log of the optimization iterations produced in phase 1 by LLMPHY. Specifically, the text shows how the LLM uses the provided feedback to adjust the physical attributes, with explanation of the justification for the adjustments. Note that each response from the LLM is separated by strings "=====

In order to produce the desired trajectories given in `problem\_trajectories.txt`, we need to adjust the physics parameters for the objects (`bottle`, `wine\_glass`, and `martini\_glass`) in the scene. Based on the instructions, we need to adjust the `sliding-friction`, `damping`, `stiffness`, and `armature` parameters. We should follow the given code structure closely and only change the parameters to their first significant digit.

Below is the adjusted Python code for `example\_code\_1.py` to produce the desired trajectories:

```

'''python
# example_code_problem.py

sim = SIMULATOR_MODEL()
sim.create_pusher('3.0 3.0 0.05')

# The physics of the tray with the ground remains the same
physical_parameters_for_object_id_tray = {
    'sliding-friction': 0.1,
    'armature': 0.1,
    'stiffness': 0.0,
    'mass': 0.5,
    'damping': 20
}
sim.create_tray(object_physics = physical_parameters_for_object_id_tray)

# Adjusted physics parameters for the bottle
physical_parameters_for_object_id_1 = {
    'sliding-friction': 0.2,
    'armature': 0.3,
    'stiffness': 0.4,
    'mass': 20.0, # mass remains unchanged
    'damping': 6.5
}
sim.create_object(object_id=1, object_name='bottle', object_location=('
row_1', 'column_3'), object_color='orange', object_physics=
physical_parameters_for_object_id_1)

# Adjusted physics parameters for the martini_glass
physical_parameters_for_object_id_2 = {
    'sliding-friction': 0.6,
    'armature': 0.5,
    'stiffness': 1.1,
    'mass': 10.0, # mass remains unchanged
    'damping': 9.0
}
sim.create_object(object_id=2, object_name='martini_glass',
object_location=('row_1', 'column_2'), object_color='orange',
object_physics=physical_parameters_for_object_id_2)

# Adjusted physics parameters for the wine_glass
physical_parameters_for_object_id_3 = {
    'sliding-friction': 0.8,
    'armature': 0.9,

```

```

1782         'stiffness': 1.0,
1783         'mass': 4.0, # mass remains unchanged
1784         'damping': 8.4
1785     }
1786     sim.create_object(object_id=3, object_name='wine_glass', object_location
1787                     =('row_1', 'column_1'), object_color='orange', object_physics=
1788                     physical_parameters_for_object_id_3)
1789
1789     sim.create_scene()
1790     sim_out = sim.run_simulation()
1791     del sim
1792     ```
1793
1793     In this adjusted code, I have modified the `sliding-friction`, `damping`,
1794     `stiffness`, and `armature` parameters for each object to produce
1795     the desired motion trajectories as suggested by the `
1796     problem_trajectories.txt` file. These modifications respect the
1797     constraint that parameters can only be adjusted up to their first
1798     significant digit.
1799
1799     You can now run this simulation code to generate the trajectories which
1800     should match closely to the ones provided in `problem_trajectories.
1801     txt`.
1802
1802     =====
1803
1804     Let's see if we can refine the physical parameters for the objects to
1805     reduce the trajectory errors. To do this, we need to adjust the
1806     parameters more carefully while adhering to the given constraints.
1807
1807     Let's analyze the trajectories and try to make minor adjustments to the
1808     parameters:
1809
1810     1. Bottle: The predicted trajectory error is 1.9, implying the need
1811     for finer adjustments. The last attempt modified the parameters to:
1812     - `sliding-friction`: 0.2
1813     - `armature`: 0.3
1814     - `stiffness`: 0.4
1815     - `damping`: 6.5
1816
1816     2. Martini_glass: The predicted trajectory error is relatively high
1817     at 3.16. We will need to make significant adjustments.
1818     - `sliding-friction`: 0.6
1819     - `armature`: 0.5
1820     - `stiffness`: 1.1
1821     - `damping`: 9.0
1822
1822     Let's reduce the damping a bit and adjust other parameters slightly.
1823
1823     3. Wine_glass: The predicted trajectory error is low at 0.24.
1824     Therefore, fine adjustments may be enough.
1825     - `sliding-friction`: 0.8
1826     - `armature`: 0.9
1827     - `stiffness`: 1.0
1828     - `damping`: 8.4
1829
1829     ### New physical parameters:
1830
1831     ```python
1832     sim = SIMULATOR_MODEL()
1833     sim.create_pusher('3.0 3.0 0.05')
1834     physical_parameters_for_object_id_tray = {
1835         'sliding-friction': 0.1,
1836         'armature': 0.1,
1837         'stiffness': 0.0,

```

```

1836     'mass': 0.5,
1837     'damping': 20
1838 }
1839
1840 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
1841
1842 physical_parameters_for_object_id_1 = {
1843     'sliding-friction': 0.25, # slight increase in friction
1844     'armature': 0.4,        # slight increase in armature
1845     'stiffness': 0.5,       # slight increase in stiffness
1846     'mass': 20.0,          # mass remains unchanged
1847     'damping': 7.0         # slight increase in damping
1848 }
1849
1850 sim.create_object(object_id=1, object_name='bottle', object_location=('
1851     row_1', 'column_3'), object_color='orange', object_physics=
1852     physical_parameters_for_object_id_1)
1853
1854 physical_parameters_for_object_id_2 = {
1855     'sliding-friction': 0.55, # slight decrease in friction
1856     'armature': 0.6,        # slight increase in armature
1857     'stiffness': 1.2,       # slight increase in stiffness
1858     'mass': 10.0,          # mass remains unchanged
1859     'damping': 8.5         # decrease in damping
1860 }
1861
1862 sim.create_object(object_id=2, object_name='martini_glass',
1863     object_location=('row_1', 'column_2'), object_color='orange',
1864     object_physics=physical_parameters_for_object_id_2)
1865
1866 physical_parameters_for_object_id_3 = {
1867     'sliding-friction': 0.75, # slight decrease in friction
1868     'armature': 1.0,        # increase in armature
1869     'stiffness': 0.9,       # slight decrease in stiffness
1870     'mass': 4.0,           # mass remains unchanged
1871     'damping': 8.2         # slight decrease in damping
1872 }
1873
1874 sim.create_object(object_id=3, object_name='wine_glass', object_location
1875     =('row_1', 'column_1'), object_color='orange', object_physics=
1876     physical_parameters_for_object_id_3)
1877
1878 sim.create_scene()
1879 sim_out=sim.run_simulation()
1880 del sim
1881 ```
1882
1883 Please use the above code to run the simulation again and see if the
1884 trajectory errors decrease. Adjustments have been made based on the
1885 analysis of the previous attempt.
1886
1887 =====
1888
1889 Based on the trend and error information from the previous attempts, we'
1890 ll refine the parameters further to attempt a better match for the
1891 trajectories in 'problem_trajectories.txt'. The key insights from
1892 previous attempts suggest that we need finer adjustments,
1893 particularly to friction, damping, and stiffness.
1894
1895 Here's an updated attempt:
1896
1897 ```python
1898 # Code structure and constants same as provided
1899
1900 sim = SIMULATOR_MODEL()

```



```

1890 sim.create_pusher('3.0 3.0 0.05')
1891 physical_parameters_for_object_id_tray = {
1892     'sliding-friction': 0.1,
1893     'armature': 0.1,
1894     'stiffness': 0.0,
1895     'mass': 0.5,
1896     'damping': 20
1897 }
1898 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
1899
1900 # Refine bottle parameters
1901 physical_parameters_for_object_id_1 = {
1902     'sliding-friction': 0.23, # slight increase in friction
1903     'armature': 0.45,      # increase in armature
1904     'stiffness': 0.55,    # slight increase in stiffness
1905     'mass': 20.0,        # mass remains unchanged
1906     'damping': 7.5       # increase in damping
1907 }
1908 sim.create_object(object_id=1, object_name='bottle', object_location=(
1909     'row_1', 'column_3'), object_color='orange', object_physics=
1910     physical_parameters_for_object_id_1)
1911
1912 # Refine martini_glass parameters
1913 physical_parameters_for_object_id_2 = {
1914     'sliding-friction': 0.54, # slight decrease in friction
1915     'armature': 0.65,      # increase in armature
1916     'stiffness': 1.25,    # increase in stiffness
1917     'mass': 10.0,        # mass remains unchanged
1918     'damping': 8.3       # slight decrease in damping
1919 }
1920 sim.create_object(object_id=2, object_name='martini_glass',
1921     object_location=('row_1', 'column_2'), object_color='orange',
1922     object_physics=physical_parameters_for_object_id_2)
1923
1924 # Refine wine_glass parameters
1925 physical_parameters_for_object_id_3 = {
1926     'sliding-friction': 0.72, # slight decrease in friction
1927     'armature': 1.05,      # increase in armature
1928     'stiffness': 0.95,    # slight decrease in stiffness
1929     'mass': 4.0,        # mass remains unchanged
1930     'damping': 8.1       # decrease in damping
1931 }
1932 sim.create_object(object_id=3, object_name='wine_glass', object_location
1933     =('row_1', 'column_1'), object_color='orange', object_physics=
1934     physical_parameters_for_object_id_3)
1935
1936 # Run the simulation
1937 sim.create_scene()
1938 sim_out = sim.run_simulation()
1939 del sim
1940 '''
1941
1942 This refinement aims to balance the dynamics and interactions of the
1943 objects with the tray, considering the error metrics from previous
1944 attempts. Monitoring the changes in each property helps to
1945 iteratively converge towards a more accurate trajectory.
1946
1947 =====
1948
1949 Based on the provided attempts and their errors, let's attempt to refine
1950 the physical parameters for the objects in such a way to specifically
1951 address the notable errors. Here's another attempt:
1952
1953 1. The bottle ('object_id=1') has a relatively high trajectory error, so
1954 we may need to adjust its parameters further to get a close match.

```

```

1944 2. The martini_glass ('object_id=2') has the highest trajectory error,
1945     requiring focus on reducing its discrepancy.
1946 3. The wine_glass ('object_id=3') has the least error, suggesting that
1947     its parameters are close to what's needed but might still need slight
1948     adjustments to further align trajectories.
1949
1950 Considering these points, we'll make small but precise adjustments,
1951     particularly targeting increased damping for the bottle, slightly
1952     decreasing friction for the martini_glass, and incrementally changing
1953     the armature and stiffness for both martini_glass and wine_glass.
1954
1955 Here is the code:
1956
1957 ```python
1958 # Generated Python code to accurately reproduce the trajectories
1959
1960 sim = SIMULATOR_MODEL()
1961 sim.create_pusher('3.0 3.0 0.05')
1962
1963 physical_parameters_for_object_id_tray = {
1964     'sliding-friction': 0.1,
1965     'armature': 0.1,
1966     'stiffness': 0.0,
1967     'mass': 0.5,
1968     'damping': 20
1969 }
1970 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
1971
1972 physical_parameters_for_object_id_1 = {
1973     'sliding-friction': 0.22, # slight decrease in friction to adjust
1974     movement
1975     'armature': 0.43, # slight decrease in armature
1976     'stiffness': 0.53, # slight decrease in stiffness
1977     'mass': 20.0, # mass remains unchanged
1978     'damping': 7.4 # slight decrease in damping
1979 }
1980 sim.create_object(object_id=1, object_name='bottle', object_location=('
1981 row_1', 'column_3'), object_color='orange', object_physics=
1982 physical_parameters_for_object_id_1)
1983
1984 physical_parameters_for_object_id_2 = {
1985     'sliding-friction': 0.52, # slight decrease in friction to reduce
1986     error
1987     'armature': 0.62, # slight decrease in armature
1988     'stiffness': 1.23, # slight decrease in stiffness
1989     'mass': 10.0, # mass remains unchanged
1990     'damping': 8.1 # slight decrease in damping to reduce discrepancy
1991 }
1992 sim.create_object(object_id=2, object_name='martini_glass',
1993 object_location=('row_1', 'column_2'), object_color='orange',
1994 object_physics=physical_parameters_for_object_id_2)
1995
1996 physical_parameters_for_object_id_3 = {
1997     'sliding-friction': 0.71, # slight decrease in friction for close
1998     alignment
1999     'armature': 1.04, # slight decrease in armature
2000     'stiffness': 0.94, # slight decrease in stiffness
2001     'mass': 4.0, # mass remains unchanged
2002     'damping': 8.0 # slight decrease in damping for close alignment
2003 }
2004 sim.create_object(object_id=3, object_name='wine_glass', object_location
2005 =('row_1', 'column_1'), object_color='orange', object_physics=
2006 physical_parameters_for_object_id_3)
2007
2008 sim.create_scene()

```

```

1998 sim_out=sim.run_simulation()
1999 del sim
2000 ```
2001
2002 Make sure to run this code through your simulation environment to verify
2003 the trajectory errors. The adjustments are subtle yet precise to
2004 better align with the required trajectories in 'problem_trajectories.
2005 txt'.
2006
2007 =====
2008
2009 To refine the parameters and reduce the trajectory prediction error, I
2010 will make gradual and strategic adjustments while making sure not to
2011 alter beyond the first significant digit. Below are the physical
2012 parameters for each object, followed by the refined code that will be
2013 used to generate the trajectories.
2014
2015 ```python
2016 # refined_code.py
2017
2018 sim = SIMULATOR_MODEL()
2019 sim.create_pusher('3.0 3.0 0.05')
2020
2021 physical_parameters_for_object_id_tray = {
2022     'sliding-friction': 0.1,
2023     'armature': 0.1,
2024     'stiffness': 0.0,
2025     'mass': 0.5,
2026     'damping': 20
2027 }
2028 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
2029
2030 physical_parameters_for_object_id_1 = {
2031     'sliding-friction': 0.21, # slight decrease in friction for finer
2032     adjustment
2033     'armature': 0.42, # slight decrease in armature
2034     'stiffness': 0.54, # slight adjustment in stiffness
2035     'mass': 20.0, # mass remains unchanged
2036     'damping': 7.3 # slight decrease in damping
2037 }
2038 sim.create_object(object_id=1, object_name='bottle', object_location=('
2039 row_1', 'column_3'), object_color='orange', object_physics=
2040 physical_parameters_for_object_id_1)
2041
2042 physical_parameters_for_object_id_2 = {
2043     'sliding-friction': 0.51, # slight decrease in friction to further
2044     reduce error
2045     'armature': 0.61, # slight decrease in armature
2046     'stiffness': 1.22, # slight adjustment in stiffness
2047     'mass': 10.0, # mass remains unchanged
2048     'damping': 8.0 # slight decrease in damping
2049 }
2050 sim.create_object(object_id=2, object_name='martini_glass',
2051 object_location=('row_1', 'column_2'), object_color='orange',
2052 object_physics=physical_parameters_for_object_id_2)
2053
2054 physical_parameters_for_object_id_3 = {
2055     'sliding-friction': 0.73, # slight increase in friction for better
2056     alignment
2057     'armature': 1.03, # slight decrease in armature
2058     'stiffness': 0.96, # slight increase in stiffness
2059     'mass': 4.0, # mass remains unchanged
2060     'damping': 7.9 # slight decrease in damping
2061 }

```

```

2052 sim.create_object(object_id=3, object_name='wine_glass', object_location
2053     =('row_1', 'column_1'), object_color='orange', object_physics=
2054     physical_parameters_for_object_id_3)
2055
2056 # Create and run the scene
2057 sim.create_scene()
2058 sim_out = sim.run_simulation()
2059 del sim
2060 ```
2061 The above adjustments are strategic, aiming for finer tuning of the
2062 sliding-friction, armature, stiffness, and damping parameters to get
2063 results closer to the desired trajectories. By changing only up to
2064 the first significant digit, these increments offer more precise
2065 control over the movements of the objects in the simulation.
2066
2067 You can run this refined code in your simulation environment to compare
2068 the actual trajectory errors and see if the refinements have reduced
2069 the prediction errors further.
2070
2071 =====
2072
2073 To refine the parameters further, let's focus on making smaller and yet
2074 significant adjustments based on the previous attempts and their
2075 respective errors. We will perform slight tweaking to the parameters
2076 to minimize the trajectory errors further.
2077
2078 Given that the previous attempts were gradually reducing errors, let's
2079 make slight fine-tuning adjustments to improve the accuracy even
2080 further:
2081
2082 - Slightly tweaking the sliding-friction for each object to balance the
2083 motion.
2084 - Small adjustments to armature, stiffness, and damping to achieve
2085 optimal results.
2086 - Keeping all changes within the first significant digit, as instructed.
2087
2088 Here is the adjusted version of the code that aims to further refine the
2089 object parameters:
2090
2091 ```python
2092 sim = SIMULATOR_MODEL()
2093 sim.create_pusher('3.0 3.0 0.05')
2094
2095 physical_parameters_for_object_id_tray = {
2096     'sliding-friction': 0.1,
2097     'armature': 0.1,
2098     'stiffness': 0.0,
2099     'mass': 0.5,
2100     'damping': 20
2101 }
2102 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
2103
2104 physical_parameters_for_object_id_1 = {
2105     'sliding-friction': 0.22, # slight increase in friction
2106     'armature': 0.41, # slight decrease in armature
2107     'stiffness': 0.52, # slight adjustment in stiffness
2108     'mass': 20.0, # mass remains unchanged
2109     'damping': 7.1 # slight decrease in damping
2110 }
2111 sim.create_object(object_id=1, object_name='bottle', object_location=('
2112     row_1', 'column_3'), object_color='orange', object_physics=
2113     physical_parameters_for_object_id_1)
2114
2115 physical_parameters_for_object_id_2 = {

```

```

2106     'sliding-friction': 0.53, # slight decrease in friction
2107     'armature': 0.64,      # slight increase in armature
2108     'stiffness': 1.24,    # slight adjustment in stiffness
2109     'mass': 10.0,        # mass remains unchanged
2110     'damping': 8.1       # slight decrease in damping
2111 }
2112 sim.create_object(object_id=2, object_name='martini_glass',
2113                  object_location=('row_1', 'column_2'), object_color='orange',
2114                  object_physics=physical_parameters_for_object_id_2)
2115 physical_parameters_for_object_id_3 = {
2116     'sliding-friction': 0.72, # slight decrease in friction
2117     'armature': 1.02,      # slight decrease in armature
2118     'stiffness': 0.93,    # slight decrease in stiffness
2119     'mass': 4.0,         # mass remains unchanged
2120     'damping': 7.8       # slight decrease in damping
2121 }
2122 sim.create_object(object_id=3, object_name='wine_glass', object_location
2123                  =('row_1', 'column_1'), object_color='orange', object_physics=
2124                  physical_parameters_for_object_id_3)
2125 sim.create_scene()
2126 sim_out = sim.run_simulation()
2127 del sim
2128 ```
2129 This adjustment attempts to balance the parameters more precisely by
2130 making small changes while focusing on the physical behavior that
2131 impacts the simulation. The goal remains to accurately reproduce the
2132 trajectories provided in 'problem_trajectories.txt'.
2133 =====
2134 To further refine the simulation parameters and achieve lower trajectory
2135 prediction errors, we will use small adjustments based on the
2136 previous attempts. Here is the updated code based on a careful
2137 analysis of the prior errors and corresponding changes:
2138
2139 ```python
2140 sim = SIMULATOR_MODEL()
2141 sim.create_pusher('3.0 3.0 0.05')
2142 physical_parameters_for_object_id_tray = {
2143     'sliding-friction': 0.1,
2144     'armature': 0.1,
2145     'stiffness': 0.0,
2146     'mass': 0.5,
2147     'damping': 20
2148 }
2149 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2150 physical_parameters_for_object_id_1 = {
2151     'sliding-friction': 0.22, # increased slightly
2152     'armature': 0.40,      # decreased slightly
2153     'stiffness': 0.53,    # adjusted slightly
2154     'mass': 20.0,        # mass remains unchanged
2155     'damping': 7.2       # decreased slightly
2156 }
2157 sim.create_object(object_id=1, object_name='bottle', object_location=(
2158     'row_1', 'column_3'), object_color='orange', object_physics=
2159     physical_parameters_for_object_id_1)
2160 physical_parameters_for_object_id_2 = {
2161     'sliding-friction': 0.54, # decreased slightly

```

```

2160     'armature': 0.63,      # adjusted slightly
2161     'stiffness': 1.23,    # adjusted slightly
2162     'mass': 10.0,        # mass remains unchanged
2163     'damping': 8.2       # adjusted slightly
2164 }
2165 sim.create_object(object_id=2, object_name='martini_glass',
2166                  object_location=('row_1', 'column_2'), object_color='orange',
2167                  object_physics=physical_parameters_for_object_id_2)
2168 physical_parameters_for_object_id_3 = {
2169     'sliding-friction': 0.73, # increased slightly
2170     'armature': 1.04,        # adjusted slightly
2171     'stiffness': 0.94,      # adjusted slightly
2172     'mass': 4.0,           # mass remains unchanged
2173     'damping': 8.0         # adjusted slightly
2174 }
2175 sim.create_object(object_id=3, object_name='wine_glass', object_location
2176                  =('row_1', 'column_1'), object_color='orange', object_physics=
2177                  physical_parameters_for_object_id_3)
2178
2179 sim.create_scene()
2180 sim_out=sim.run_simulation()
2181 del sim
2182 ```
2183
2184 Filename: `example_code_refined.py`
2185
2186 Explanation of Changes:
2187 1. **Bottle (object_id_1)**:
2188     - Increased sliding friction from 0.21 to 0.22 to further adjust
2189       movement.
2190     - Decreased armature from 0.42 to 0.40 for a finer balance.
2191     - Adjusted stiffness slightly from 0.54 to 0.53.
2192     - Decreased damping from 7.3 to 7.2 for finer tuning.
2193
2194 2. **Martini Glass (object_id_2)**:
2195     - Decreased sliding friction to 0.54 for slight adjustment.
2196     - Increased armature slightly from 0.61 to 0.63.
2197     - Adjusted stiffness slightly to 1.23.
2198     - Decreased damping to 8.2 for finer tuning.
2199
2200 3. **Wine Glass (object_id_3)**:
2201     - Increased sliding friction slightly to 0.73 for better alignment.
2202     - Adjusted armature to 1.04 for balance.
2203     - Adjusted stiffness to 0.94.
2204     - Kept damping at 8.0 for slight adjustment.
2205
2206 This should result in a better alignment of the predicted trajectories
2207 with the ones given in `problem_trajectories.txt`.
2208
2209 =====
2210
2211 To refine the physical parameters further and reduce the trajectory
2212 prediction errors, let's analyze the errors from previous attempts
2213 and try to make incremental adjustments.
2214
2215 Given the previous errors and the adjustments made, it is noticed that
2216 attempts have incrementally changed both the sliding-friction and the
2217 armature values slightly, while trying to balance out the
2218 trajectories of the bottle, martini_glass, and wine_glass.
2219
2220 Let's make further fine-tuned adjustments to these values in an effort to
2221 bring the prediction errors closer to the intended trajectories.
2222
2223 ```python

```



```

2214 sim = SIMULATOR_MODEL()
2215 sim.create_pusher('3.0 3.0 0.05')
2216 physical_parameters_for_object_id_tray = {
2217     'sliding-friction': 0.1,
2218     'armature': 0.1,
2219     'stiffness': 0.0,
2220     'mass': 0.5,
2221     'damping': 20
2222 }
2223 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2224 physical_parameters_for_object_id_1 = {
2225     'sliding-friction': 0.23, # slight increase
2226     'armature': 0.39, # slight decrease
2227     'stiffness': 0.53, # slight adjustment
2228     'mass': 20.0, # mass remains unchanged
2229     'damping': 7.2 # slight adjustment
2230 }
2231 sim.create_object(object_id=1, object_name='bottle', object_location=('
2232     row_1', 'column_3'), object_color='orange', object_physics=
2233     physical_parameters_for_object_id_1)
2234 physical_parameters_for_object_id_2 = {
2235     'sliding-friction': 0.53, # decrease slightly
2236     'armature': 0.66, # increase slightly
2237     'stiffness': 1.23, # adjusted slightly
2238     'mass': 10.0, # mass remains unchanged
2239     'damping': 8.2 # slight adjustment
2240 }
2241 sim.create_object(object_id=2, object_name='martini_glass',
2242     object_location=('row_1', 'column_2'), object_color='orange',
2243     object_physics=physical_parameters_for_object_id_2)
2244 physical_parameters_for_object_id_3 = {
2245     'sliding-friction': 0.72, # slight decrease
2246     'armature': 1.05, # slight increase
2247     'stiffness': 0.96, # slight adjustment
2248     'mass': 4.0, # mass remains unchanged
2249     'damping': 7.9 # slight adjustment
2250 }
2251 sim.create_object(object_id=3, object_name='wine_glass', object_location
2252     =('row_1', 'column_1'), object_color='orange', object_physics=
2253     physical_parameters_for_object_id_3)
2254 del sim
2255 ```
2256 This updated code aims to fine-tune the physical parameters (sliding-
2257     friction, armature, stiffness, and damping) based on the errors
2258     observed in previous attempts. The adjustments aim for minimal yet
2259     effective changes to reduce the trajectory prediction errors. You
2260     should use this code and verify the resulting trajectory errors to
2261     see if they are minimized.
2262     =====
2263     Below is a refined code structure to simulate the scene, taking into
2264     consideration the errors from previous attempts and aiming to
2265     minimize the overall trajectory prediction error. The goal is to fine
2266     -tune the physical parameters for each object to closely match the
2267     provided `problem_trajectories.txt`.

```

```

2268   ```python
2269   # refined_code.py
2270
2271   sim = SIMULATOR_MODEL()
2272   sim.create_pusher('3.0 3.0 0.05')
2273
2274   # Tray parameters remain unchanged as per the instructions
2275   physical_parameters_for_tray = {
2276       'sliding-friction': 0.1,
2277       'armature': 0.1,
2278       'stiffness': 0.0,
2279       'mass': 0.5,
2280       'damping': 20
2281   }
2282   sim.create_tray(object_physics=physical_parameters_for_tray)
2283
2284   # New calculated parameters for bottle
2285   physical_parameters_for_object_id_1 = {
2286       'sliding-friction': 0.22, # slight decrease in friction for finer
2287       adjustment
2288       'armature': 0.42, # slight adjustment in armature
2289       'stiffness': 0.54, # slight increase in stiffness
2290       'mass': 20.0, # mass remains unchanged
2291       'damping': 7.3 # slight decrease in damping
2292   }
2293   sim.create_object(object_id=1, object_name='bottle', object_location=('
2294   row_1', 'column_3'), object_color='orange', object_physics=
2295   physical_parameters_for_object_id_1)
2296
2297   # New calculated parameters for martini_glass
2298   physical_parameters_for_object_id_2 = {
2299       'sliding-friction': 0.51, # slight decrease in friction for reduction
2300       in error
2301       'armature': 0.61, # adjusted slightly
2302       'stiffness': 1.21, # adjusted slightly
2303       'mass': 10.0, # mass remains unchanged
2304       'damping': 8.0 # adjusted slightly
2305   }
2306   sim.create_object(object_id=2, object_name='martini_glass',
2307   object_location=('row_1', 'column_2'), object_color='orange',
2308   object_physics=physical_parameters_for_object_id_2)
2309
2310   # New calculated parameters for wine_glass
2311   physical_parameters_for_object_id_3 = {
2312       'sliding-friction': 0.73, # adjusted for better alignment
2313       'armature': 1.03, # slight adjustment
2314       'stiffness': 0.94, # slight increase in stiffness
2315       'mass': 4.0, # mass remains unchanged
2316       'damping': 7.9 # slight adjustment
2317   }
2318   sim.create_object(object_id=3, object_name='wine_glass', object_location
2319   =('row_1', 'column_1'), object_color='orange', object_physics=
2320   physical_parameters_for_object_id_3)
2321
2322   sim.create_scene()
2323   sim_out = sim.run_simulation()
2324   del sim
2325   ```
2326
2327   ### Key Adjustments Made:
2328   - **Bottle (ID 1)**:
2329     - Slight decrease in sliding friction to 0.22.
2330     - Adjusted armature to 0.42.
2331     - Increased stiffness to 0.54.
2332     - Decreased damping to 7.3.

```

```

2322
2323 - **Martini Glass (ID 2)**:
2324   - Decreased sliding friction to 0.51.
2325   - Adjusted armature to 0.61.
2326   - Stiffness set to 1.21.
2327   - Damping decreased to 8.0.
2328
2329 - **Wine Glass (ID 3)**:
2330   - Sliding friction adjusted up to 0.73.
2331   - Armature adjusted to 1.03.
2332   - Stiffness increased to 0.94.
2333   - Damping adjusted to 7.9.
2334
2335 These adjustments target reducing the overall prediction error based on
2336 the last feedback received. Running this simulation with these
2337 parameters should yield predictions closer to the proposed `
2338 problem_trajectories.txt`.
2339
2340 =====
2341
2342 To further refine the physical parameters and reduce the trajectory
2343 prediction error, let's carefully adjust each parameter based on the
2344 insights from previous attempts. It's crucial to keep the adjustments
2345 minimal and focus on specific discrepancies between the predicted
2346 and actual trajectories to fine-tune the parameters more effectively.
2347
2348 Here is the Python code for `example_code_1.py` with updated physical
2349 parameters:
2350
2351 ```python
2352 sim = SIMULATOR_MODEL()
2353 sim.create_pusher('3.0 3.0 0.05')
2354
2355 # Physics parameters for the tray (unchanged)
2356 physical_parameters_for_object_id_tray = {
2357     'sliding-friction': 0.1,
2358     'armature': 0.1,
2359     'stiffness': 0.0,
2360     'mass': 0.5,
2361     'damping': 20
2362 }
2363 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2364
2365 # Adjusted physical parameters for the bottle
2366 physical_parameters_for_object_id_1 = {
2367     'sliding-friction': 0.21, # slight decrease
2368     'armature': 0.43, # slight adjustment
2369     'stiffness': 0.53, # slight adjustment
2370     'mass': 20.0, # mass remains unchanged
2371     'damping': 7.2 # slight adjustment
2372 }
2373 sim.create_object(object_id=1, object_name='bottle', object_location=('
2374 row_1', 'column_3'), object_color='orange', object_physics=
2375 physical_parameters_for_object_id_1)
2376
2377 # Adjusted physical parameters for the martini_glass
2378 physical_parameters_for_object_id_2 = {
2379     'sliding-friction': 0.52, # slight adjustment
2380     'armature': 0.64, # slight adjustment
2381     'stiffness': 1.21, # slight adjustment
2382     'mass': 10.0, # mass remains unchanged
2383     'damping': 8.1 # slight adjustment
2384 }
2385

```

```

2376 sim.create_object(object_id=2, object_name='martini_glass',
2377                  object_location=('row_1', 'column_2'), object_color='orange',
2378                  object_physics=physical_parameters_for_object_id_2)
2379
2380 # Adjusted physical parameters for the wine_glass
2381 physical_parameters_for_object_id_3 = {
2382     'sliding-friction': 0.73, # slight adjustment
2383     'armature': 1.03, # slight adjustment
2384     'stiffness': 0.93, # slight adjustment
2385     'mass': 4.0, # mass remains unchanged
2386     'damping': 7.8 # slight adjustment
2387 }
2388 sim.create_object(object_id=3, object_name='wine_glass', object_location
2389                  =('row_1', 'column_1'), object_color='orange', object_physics=
2390                  physical_parameters_for_object_id_3)
2391
2392 sim.create_scene()
2393 sim_out = sim.run_simulation()
2394 del sim
2395 ```
2396
2397 Explanation of refinement:
2398 1. Sliding Friction: Slightly decreased for the bottle and
2399 martini_glass based on their sensitivity.
2400 2. Armature: Tweaked by a small degree for each object considering
2401 their resistance to movement.
2402 3. Stiffness: Adjusted to fine-tune the object's resistance to
2403 deformation.
2404 4. Damping: Decreased slightly to improve the motion accuracy of all
2405 objects.
2406
2407 This fine-tuned approach aims to reduce the prediction error while
2408 ensuring minimal changes to the parameters, focusing on enhancing the
2409 alignment of the trajectories with the given problem_trajectories.
2410 txt. Adjust only to their first significant digit to conform to the
2411 constraints.
2412
2413 =====
2414
2415 Given the previous attempts and their errors, it seems that the
2416 adjustments have incrementally improved but not drastically reduced
2417 the errors. To reduce errors further, we should continue fine-tuning
2418 the parameters while considering the inherent dynamics of each object
2419 . Here is another attempt with refined parameters:
2420
2421 ```python
2422 # example_code_1.py
2423
2424 sim = SIMULATOR_MODEL()
2425 sim.create_pusher('3.0 3.0 0.05')
2426 physical_parameters_for_object_id_tray = {
2427     'sliding-friction': 0.1,
2428     'armature': 0.1,
2429     'stiffness': 0.0,
2430     'mass': 0.5,
2431     'damping': 20
2432 }
2433 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2434 physical_parameters_for_object_id_1 = {
2435     'sliding-friction': 0.21, # slight decrease for finer control
2436     'armature': 0.41, # adjusted armature
2437     'stiffness': 0.55, # slight increase in stiffness
2438     'mass': 20.0, # mass remains unchanged
2439     'damping': 7.0 # slight decrease in damping
2440 }

```

```

2430 sim.create_object(object_id=1, object_name='bottle', object_location=('
2431 row_1', 'column_3'), object_color='orange', object_physics=
2432 physical_parameters_for_object_id_1)
2433
2434 physical_parameters_for_object_id_2 = {
2435     'sliding-friction': 0.52, # adjusted friction
2436     'armature': 0.63, # slight adjustment in armature
2437     'stiffness': 1.22, # slight adjustment in stiffness
2438     'mass': 10.0, # mass remains unchanged
2439     'damping': 8.0 # slight increase in damping
2440 }
2441 sim.create_object(object_id=2, object_name='martini_glass',
2442 object_location=('row_1', 'column_2'), object_color='orange',
2443 object_physics=physical_parameters_for_object_id_2)
2444
2445 physical_parameters_for_object_id_3 = {
2446     'sliding-friction': 0.72, # slight adjustment
2447     'armature': 1.02, # slight decrease
2448     'stiffness': 0.94, # slight adjustment in stiffness
2449     'mass': 4.0, # mass remains unchanged
2450     'damping': 7.9 # slight adjustment
2451 }
2452 sim.create_object(object_id=3, object_name='wine_glass', object_location
2453 =('row_1', 'column_1'), object_color='orange', object_physics=
2454 physical_parameters_for_object_id_3)
2455
2456 sim.create_scene()
2457 sim_out = sim.run_simulation()
2458 del sim
2459 ```
2460
2461 ### Key Adjustments:
2462 1. **Bottle (`object_id=1`):**
2463 - Reduced sliding friction further to improve response.
2464 - Increased stiffness for better stability.
2465 - Slightly reduced damping to allow more movement.
2466
2467 2. **Martini Glass (`object_id=2`):**
2468 - Adjusted sliding friction and armature for more controlled dynamics.
2469 - Slightly increased damping to prevent overshooting.
2470 - Marginal adjustments in stiffness.
2471
2472 3. **Wine Glass (`object_id=3`):**
2473 - Slightly adjusted sliding friction and armature for finer tuning.
2474 - Minor adjustments in stiffness for better alignment.
2475 - Damping adjusted for controlled oscillations.
2476
2477 Run this updated code in your simulator and check if the error is further
2478 reduced. It's crucial to analyze the trajectories post-simulation to
2479 ensure each parameter adjustment is aiding in convergence toward the
2480 desired trajectories.
2481
2482 =====
2483
2484 To optimize the code further and reduce the trajectory error, I'll base
2485 my changes on the closest attempts with slight adjustments to each
2486 parameter to improve the alignment with 'problem_trajectories.txt'.
2487
2488 Let's try small incremental adjustments of the successful parameters from
2489 GPT Attempt #11:
2490
2491 ```python
2492 # example_code_optimized.py
2493
2494 sim = SIMULATOR_MODEL()

```

```

2484 sim.create_pusher('3.0 3.0 0.05')
2485
2486 # Tray parameters remain unchanged
2487 physical_parameters_for_object_id_tray = {
2488     'sliding-friction': 0.1,
2489     'armature': 0.1,
2490     'stiffness': 0.0,
2491     'mass': 0.5,
2492     'damping': 20
2493 }
2494 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2495
2496 # Optimized parameters based on closest previous attempt with slight
2497 adjustments
2498 physical_parameters_for_object_id_1 = { # bottle
2499     'sliding-friction': 0.22, # slight increase for better alignment
2500     'armature': 0.42, # slight adjustment
2501     'stiffness': 0.54, # slight adjustment
2502     'mass': 20.0, # mass remains unchanged
2503     'damping': 7.1 # slight adjustment
2504 }
2505
2506 physical_parameters_for_object_id_2 = { # martini_glass
2507     'sliding-friction': 0.53, # slight adjustment
2508     'armature': 0.64, # slight adjustment
2509     'stiffness': 1.23, # slight adjustment
2510     'mass': 10.0, # mass remains unchanged
2511     'damping': 8.2 # slight adjustment
2512 }
2513
2514 physical_parameters_for_object_id_3 = { # wine_glass
2515     'sliding-friction': 0.73, # slight adjustment
2516     'armature': 1.03, # slight adjustment
2517     'stiffness': 0.95, # slight adjustment
2518     'mass': 4.0, # mass remains unchanged
2519     'damping': 7.8 # slight adjustment
2520 }
2521
2522 sim.create_object(object_id=1, object_name='bottle', object_location=('
2523 row_1', 'column_3'), object_color='orange', object_physics=
2524 physical_parameters_for_object_id_1)
2525 sim.create_object(object_id=2, object_name='martini_glass',
2526 object_location=('row_1', 'column_2'), object_color='orange',
2527 object_physics=physical_parameters_for_object_id_2)
2528 sim.create_object(object_id=3, object_name='wine_glass', object_location
2529 =('row_1', 'column_1'), object_color='orange', object_physics=
2530 physical_parameters_for_object_id_3)
2531
2532 sim.create_scene()
2533 sim_out = sim.run_simulation()
2534 del sim
2535
2536 ```
2537
2538 In this adjustment, I've increased or decreased parameters slightly
2539 within the first significant digit and maintained the structure of
2540 the initial code. This approach should bring the predicted
2541 trajectories closer in alignment with the ones in '
2542 problem_trajectories.txt'.
2543
2544 =====
2545
2546 Given the existing attempts and their respective trajectory prediction
2547 errors, let's incrementally adjust the physical parameters to better
2548 align with the desired trajectories in 'problem_trajectories.txt'. We

```

```

2538     'll focus on making small adjustments to the individual parameters to
2539     minimize the error.
2540
2541 ### Revised Code
2542
2543 ```python
2544 sim = SIMULATOR_MODEL()
2545 sim.create_pusher('3.0 3.0 0.05')
2546
2547 physical_parameters_for_object_id_tray = {
2548     'sliding-friction': 0.1,
2549     'armature': 0.1,
2550     'stiffness': 0.0,
2551     'mass': 0.5,
2552     'damping': 20
2553 }
2554 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
2555
2556 physical_parameters_for_object_id_1 = {
2557     'sliding-friction': 0.21, # decreased friction
2558     'armature': 0.41, # slightly decreased armature
2559     'stiffness': 0.54, # slight increase in stiffness
2560     'mass': 20.0, # mass remains unchanged
2561     'damping': 7.2 # slightly decreased damping
2562 }
2563 sim.create_object(object_id=1, object_name='bottle', object_location=('
2564     row_1', 'column_3'), object_color='orange', object_physics=
2565     physical_parameters_for_object_id_1)
2566
2567 physical_parameters_for_object_id_2 = {
2568     'sliding-friction': 0.52, # slightly decreased friction
2569     'armature': 0.62, # slightly decreased armature
2570     'stiffness': 1.24, # slight increase in stiffness
2571     'mass': 10.0, # mass remains unchanged
2572     'damping': 8.0 # slight increase in damping
2573 }
2574 sim.create_object(object_id=2, object_name='martini_glass',
2575     object_location=('row_1', 'column_2'), object_color='orange',
2576     object_physics=physical_parameters_for_object_id_2)
2577
2578 physical_parameters_for_object_id_3 = {
2579     'sliding-friction': 0.73, # slight increase in friction
2580     'armature': 1.02, # slightly decreased armature
2581     'stiffness': 0.95, # slight increase in stiffness
2582     'mass': 4.0, # mass remains unchanged
2583     'damping': 7.9 # slight increase in damping
2584 }
2585 sim.create_object(object_id=3, object_name='wine_glass', object_location
2586     =('row_1', 'column_1'), object_color='orange', object_physics=
2587     physical_parameters_for_object_id_3)
2588
2589 sim.create_scene()
2590 sim_out=sim.run_simulation()
2591 del sim
2592 ```

```

The changes applied here are small and precise:

- For the 'bottle':
  - Minor decrease in 'sliding-friction' to possibly tune finer control over motion.
  - Slight decrease in 'armature' for more conservative moment inertia.
  - Small increase in 'stiffness' to ensure some adjustment in impact response.
  - Slight decrease in 'damping' to adjust energy dissipation.



2592

2593 2. For the `martini\_glass`:

2594 - Slight decrease in `sliding-friction` to reduce resistance.

2595 - Slight decrease in `armature` to have fewer moments of inertia.

2596 - Small increase in `stiffness` to adjust the rigidity.

2597 - Slight increase in `damping` to tune the damping effect.

2598

2599 3. For the `wine\_glass`:

2600 - Slight increase in `sliding-friction` to increase motion resistance

2601 towards alignment.

2602 - Slight decrease in `armature` for finer inertia control.

2603 - Small increase in `stiffness` for more precise impact response.

2604 - Slight increase in `damping` for tuned energy dissipation.

2604 By maintaining the careful balance between these suggested adjustments,

2605 we are likely to achieve a closer match to the desired trajectories

2606 in `problem\_trajectories.txt`. The actual prediction should be

2607 evaluated by running this updated code in the provided simulator and

2608 examining the resulting trajectory errors.

2609 =====

2610

2611 Based on the trajectory prediction errors from previous attempts, it

2612 appears that the parameters for the `bottle` are quite close to

2613 accurate, whereas the `martini\_glass` exhibits higher errors which

2614 need to be reduced. The `wine\_glass` parameters seem to be nearly

2615 accurate.

2616 To further reduce the trajectory prediction error, we will optimize the

2617 parameters while maintaining significant digits:

2618

2619 1. Slightly adjust the `sliding-friction` and `damping` for all objects.

2620 2. Fine-tune `armature` and `stiffness` precisely for each object to

2621 better align with the desired trajectories.

2622 Here is a suggested set of parameters for another attempt:

2623 ```python

2624 sim = SIMULATOR\_MODEL()

2625 sim.create\_pusher('3.0 3.0 0.05')

2626 physical\_parameters\_for\_object\_id\_tray = {

2627 'sliding-friction': 0.1,

2628 'armature': 0.1,

2629 'stiffness': 0.0,

2630 'mass': 0.5,

2631 'damping': 20

2632 }

2633 sim.create\_tray(object\_physics = physical\_parameters\_for\_object\_id\_tray)

2634 physical\_parameters\_for\_object\_id\_1 = {

2635 'sliding-friction': 0.21, # slight adjustment in friction

2636 'armature': 0.42, # slight decrease in armature

2637 'stiffness': 0.53, # slight decrease in stiffness

2638 'mass': 20.0, # mass remains unchanged

2639 'damping': 7.3 # slight decrease in damping

2640 }

2641 physical\_parameters\_for\_object\_id\_2 = {

2642 'sliding-friction': 0.54, # slight adjustment to decrease friction

2643 'armature': 0.63, # slight adjustment

2644 'stiffness': 1.24, # slight adjustment

2645 'mass': 10.0, # mass remains unchanged

2646 'damping': 8.1 # slight adjustment

2647 }

```

2646 physical_parameters_for_object_id_3 = {
2647     'sliding-friction': 0.74, # slight increase to better align friction
2648     'armature': 1.01,      # slight adjustment
2649     'stiffness': 0.94,    # slight decrease in stiffness
2650     'mass': 4.0,         # mass remains unchanged
2651     'damping': 7.8       # slight decrease in damping
2652 }
2653
2654 sim.create_object(object_id=1, object_name='bottle', object_location=('
2655     row_1', 'column_3'), object_color='orange', object_physics=
2656     physical_parameters_for_object_id_1)
2657 sim.create_object(object_id=2, object_name='martini_glass',
2658     object_location=('row_1', 'column_2'), object_color='orange',
2659     object_physics=physical_parameters_for_object_id_2)
2660 sim.create_object(object_id=3, object_name='wine_glass', object_location
2661     =('row_1', 'column_1'), object_color='orange', object_physics=
2662     physical_parameters_for_object_id_3)
2663
2664 sim.create_scene()
2665 sim_out=sim.run_simulation()
2666 del sim
2667 ```
2668
2669 This adjustment continues to follow the structure of the example Python
2670 code, and the changes made to the parameters are conservative to
2671 prevent large deviations while aiming to reduce the overall
2672 trajectory error. This should produce a better alignment of the
2673 trajectories to the given data in `problem_trajectories.txt`.
2674
2675 =====
2676
2677 To better align the trajectories with those in `problem_trajectories.txt
2678 ', I will refine the physical parameters for the objects. Based on
2679 previous attempts and trajectory errors, I will focus on minor
2680 adjustments, especially honing in on the slight variations in
2681 friction, armature, stiffness, and damping.
2682
2683 Here's the revised code following the given structure:
2684
2685 ```python
2686 sim = SIMULATOR_MODEL()
2687 sim.create_pusher('3.0 3.0 0.05')
2688 physical_parameters_for_object_id_tray = {
2689     'sliding-friction': 0.1,
2690     'armature': 0.1,
2691     'stiffness': 0.0,
2692     'mass': 0.5,
2693     'damping': 20
2694 }
2695 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2696
2697 physical_parameters_for_object_id_1 = { # bottle
2698     'sliding-friction': 0.22, # slight adjustment
2699     'armature': 0.42,      # fine-tune armature
2700     'stiffness': 0.54,    # fine-tune stiffness
2701     'mass': 20.0,        # mass remains unchanged
2702     'damping': 7.1       # minor adjustment
2703 }
2704 sim.create_object(object_id=1, object_name='bottle', object_location=('
2705     row_1', 'column_3'), object_color='orange', object_physics=
2706     physical_parameters_for_object_id_1)
2707
2708 physical_parameters_for_object_id_2 = { # martini_glass
2709     'sliding-friction': 0.52, # slight adjustment
2710     'armature': 0.63,      # fine-tune armature

```

```

2700     'stiffness': 1.23, # fine-tune stiffness
2701     'mass': 10.0,    # mass remains unchanged
2702     'damping': 8.1   # minor adjustment
2703 }
2704 sim.create_object(object_id=2, object_name='martini_glass',
2705                  object_location=('row_1', 'column_2'), object_color='orange',
2706                  object_physics=physical_parameters_for_object_id_2)
2707 physical_parameters_for_object_id_3 = { # wine_glass
2708     'sliding-friction': 0.73, # slight adjustment
2709     'armature': 1.04, # fine-tune armature
2710     'stiffness': 0.95, # fine-tune stiffness
2711     'mass': 4.0, # mass remains unchanged
2712     'damping': 7.8 # minor adjustment
2713 }
2714 sim.create_object(object_id=3, object_name='wine_glass', object_location
2715                  =('row_1', 'column_1'), object_color='orange', object_physics=
2716                  physical_parameters_for_object_id_3)
2717 sim.create_scene()
2718 sim_out = sim.run_simulation()
2719 del sim
2720 ```
2721 Explanation of changes:
2722 - For the bottle, martini_glass, and wine_glass, slight adjustments are
2723   made to the sliding-friction, armature, and damping parameters in
2724   very small increments to better mimic the trajectories.
2725 - The fine-tuning is based on a better alignment seen in the errors of
2726   previous attempts.
2727 Please run this refined code in your simulator to evaluate the accuracy
2728   of the trajectories produced.
2729 =====
2730 Based on the previous attempts and their resulting errors, it appears
2731   that fine-tuning the physical parameters can potentially bring the
2732   trajectory prediction error closer to the desired trajectories. Given
2733   that previous attempts were close but required further refinement,
2734   let's slightly adjust the parameters once more to seek better
2735   alignment.
2736 Here is the refined attempt:
2737 ```python
2738 sim = SIMULATOR_MODEL()
2739 sim.create_pusher('3.0 3.0 0.05')
2740
2741 physical_parameters_for_object_id_tray = {
2742     'sliding-friction': 0.1,
2743     'armature': 0.1,
2744     'stiffness': 0.0,
2745     'mass': 0.5,
2746     'damping': 20
2747 }
2748 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
2749
2750 physical_parameters_for_object_id_1 = { # bottle
2751     'sliding-friction': 0.21, # slight adjustment
2752     'armature': 0.40, # fine-tune armature
2753     'stiffness': 0.54, # fine-tune stiffness
2754     'mass': 20.0, # mass remains unchanged
2755     'damping': 7.2 # minor adjustment
2756 }

```

```

2754
2755 physical_parameters_for_object_id_2 = { # martini_glass
2756     'sliding-friction': 0.52, # slight adjustment
2757     'armature': 0.64, # fine-tune armature
2758     'stiffness': 1.23, # fine-tune stiffness
2759     'mass': 10.0, # mass remains unchanged
2760     'damping': 8.1 # minor adjustment
2761 }
2762 physical_parameters_for_object_id_3 = { # wine_glass
2763     'sliding-friction': 0.73, # slight adjustment
2764     'armature': 1.04, # fine-tune armature
2765     'stiffness': 0.94, # fine-tune stiffness
2766     'mass': 4.0, # mass remains unchanged
2767     'damping': 7.9 # minor adjustment
2768 }
2769 sim.create_object(object_id=1, object_name='bottle', object_location=('
2770     row_1', 'column_3'), object_color='orange', object_physics=
2771     physical_parameters_for_object_id_1)
2772 sim.create_object(object_id=2, object_name='martini_glass',
2773     object_location=('row_1', 'column_2'), object_color='orange',
2774     object_physics=physical_parameters_for_object_id_2)
2775 sim.create_object(object_id=3, object_name='wine_glass', object_location
2776     =('row_1', 'column_1'), object_color='orange', object_physics=
2777     physical_parameters_for_object_id_3)
2778
2779 sim.create_scene()
2780 sim_out = sim.run_simulation()
2781 del sim
2782 ```
2783
2784 This code has been adjusted based on the trend observed in previous
2785 attempts. Small refinements are made to the parameters to attempt a
2786 better match to the target trajectories:
2787
2788 1. Slight adjustments in sliding friction, armature, stiffness, and
2789 damping for each object.
2790 2. Ensured to keep all changes within the realm of minor adjustments,
2791 considering past errors.
2792
2793 These changes are aimed to further fine-tune the physics parameters
2794 closer to the desired trajectories. This should help in reducing the
2795 trajectory prediction error further.
2796
2797 =====
2798
2799 It seems like all the example attempts produced code that follows the
2800 original problem's constraints and shows some progress in reducing
2801 the trajectory error. However, there is still room for improvement in
2802 reducing the trajectory prediction error further. Let's adjust the
2803 parameters again slightly, being mindful of not changing them beyond
2804 the first significant digit, and pick a set that should help us
2805 achieve better accuracy:
2806
2807 Here's an updated set of parameters for the objects:
2808
2809 ```python
2810 sim = SIMULATOR_MODEL()
2811 sim.create_pusher('3.0 3.0 0.05')
2812 physical_parameters_for_object_id_tray = {
2813     'sliding-friction': 0.1,
2814     'armature': 0.1,
2815     'stiffness': 0.0,
2816     'mass': 0.5,

```

```

2808     'damping': 20
2809 }
2810 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
2811
2812 physical_parameters_for_object_id_1 = { # bottle
2813     'sliding-friction': 0.21, # slight adjustment in friction
2814     'armature': 0.40,      # slight decrease in armature
2815     'stiffness': 0.55,    # slight increase in stiffness
2816     'mass': 20.0,        # mass remains unchanged
2817     'damping': 7.2       # slight decrease in damping
2818 }
2819 physical_parameters_for_object_id_2 = { # martini_glass
2820     'sliding-friction': 0.53, # slight adjustment to decrease friction
2821     'armature': 0.63,      # slight adjustment in armature
2822     'stiffness': 1.22,    # slight increase in stiffness
2823     'mass': 10.0,        # mass remains unchanged
2824     'damping': 8.2       # slight decrease in damping
2825 }
2826 physical_parameters_for_object_id_3 = { # wine_glass
2827     'sliding-friction': 0.72, # slight adjustment to increase traction
2828     'armature': 1.04,      # slight decrease
2829     'stiffness': 0.96,    # slight adjustment in stiffness
2830     'mass': 4.0,         # mass remains unchanged
2831     'damping': 8.0       # slight decrease in damping
2832 }
2833 sim.create_object(object_id=1, object_name='bottle', object_location=(
2834     'row_1', 'column_3'), object_color='orange', object_physics=
2835     physical_parameters_for_object_id_1)
2836 sim.create_object(object_id=2, object_name='martini_glass',
2837     object_location=('row_1', 'column_2'), object_color='orange',
2838     object_physics=physical_parameters_for_object_id_2)
2839 sim.create_object(object_id=3, object_name='wine_glass', object_location
2840     =('row_1', 'column_1'), object_color='orange', object_physics=
2841     physical_parameters_for_object_id_3)
2842
2843 sim.create_scene()
2844 sim_out = sim.run_simulation()
2845 del sim
2846 ```
2847
2848 In this iteration:
2849 - We fine-tuned the 'sliding-friction', 'armature', 'stiffness', and '
2850 damping' values for each object, but ensured they remained within
2851 their first significant digit constraints.
2852 - We used previous attempts' values as a base to make minute adjustments
2853 that could better align the trajectories with the given problem.
2854
2855 Once this new code set is tested, it should ideally show an improvement
2856 in error reduction. Please run this updated code using your simulator
2857 environment to verify the trajectory errors. If further adjustment
2858 is needed, making iterative small changes around these values can
2859 help further fine-tune and minimize errors.
2860
2861 =====
2862
2863 To refine your code based on the trajectory prediction error results and
2864 better align with the trajectories given in 'problem_trajectories.txt
2865 ', we need to follow a reasonable approach to adjust the physical
2866 parameters for each object slightly further. Below is a refined
2867 attempt based on the provided feedback:
2868
2869 ### Refined Physical Parameters Code:

```

```

2862   ```python
2863   # example_code_1.py
2864
2865   sim = SIMULATOR_MODEL()
2866   sim.create_pusher('3.0 3.0 0.05')
2867   physical_parameters_for_object_id_tray = {
2868       'sliding-friction': 0.1,
2869       'armature': 0.1,
2870       'stiffness': 0.0,
2871       'mass': 0.5,
2872       'damping': 20
2873   }
2874   sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2875   physical_parameters_for_object_id_1 = { # bottle
2876       'sliding-friction': 0.22, # slight increase for better alignment
2877       'armature': 0.41, # fine-tune armature
2878       'stiffness': 0.55, # fine-tune stiffness a bit more
2879       'mass': 20.0, # mass remains unchanged
2880       'damping': 7.1 # slight adjustment
2881   }
2882   sim.create_object(object_id=1, object_name='bottle', object_location=('
2883       row_1', 'column_3'), object_color='orange', object_physics=
2884       physical_parameters_for_object_id_1)
2885
2886   physical_parameters_for_object_id_2 = { # martini_glass
2887       'sliding-friction': 0.52, # slight decrease for alignment
2888       'armature': 0.62, # slight fine-tune
2889       'stiffness': 1.21, # slight fine-tune
2890       'mass': 10.0, # mass remains unchanged
2891       'damping': 8.1 # slight fine-tune
2892   }
2893   sim.create_object(object_id=2, object_name='martini_glass',
2894       object_location=('row_1', 'column_2'), object_color='orange',
2895       object_physics=physical_parameters_for_object_id_2)
2896
2897   physical_parameters_for_object_id_3 = { # wine_glass
2898       'sliding-friction': 0.73, # increase slightly for better traction
2899       'armature': 1.03, # adjusted slightly
2900       'stiffness': 0.95, # fine-tune to keep closer to expected
2901       'mass': 4.0, # mass remains unchanged
2902       'damping': 8.0 # fine-tune
2903   }
2904   sim.create_object(object_id=3, object_name='wine_glass', object_location
2905       =('row_1', 'column_1'), object_color='orange', object_physics=
2906       physical_parameters_for_object_id_3)
2907
2908   demonstration_tray_motion_trajectory = [
2909       (0.0, 0.0, 0.1), (-0.7, -0.7, 0.1), (-1.2, -1.2, 0.1), (-1.6, -1.6,
2910           0.1),
2911       (-1.9, -2.0, 0.1), (-2.1, -2.3, 0.1), (-2.3, -2.5, 0.1), (-2.4, -2.6,
2912           0.1),
2913       (-2.6, -2.8, 0.1)
2914   ]
2915
2916   sim.create_scene()
2917   sim_out = sim.run_simulation()
2918   del sim
2919   ```
2920
2921   This refinement involves slight increments and decrements on the physical
2922   parameters of the objects based on the significant aberrations
2923   observed in the previous outputs. The adjustments focus on enhancing
2924   the sliding friction, armature, and damping more accurately towards
2925   the expected trajectories per the 'problem_trajectories.txt'.

```

```

2916 Ensure that you test this code using the same simulator constraints you
2917     have been using. If further fine-tuning is necessary, keep iterating
2918     with slight adjustments, focusing on realigning the most anomalous
2919     trajectory points first.
2920
2921 =====
2922
2923 Given the simulation errors and the iterations you provided, it looks
2924 like adjustments to `sliding-friction`, `armature`, `stiffness`, and
2925 `damping` were evaluated. The challenge is to find a balance among
2926 these parameters to minimize the total trajectory error. Below, I've
2927 made refined adjustments based on the observed errors in previous
2928 attempts:
2929
2930 ```python
2931 sim = SIMULATOR_MODEL()
2932 sim.create_pusher('3.0 3.0 0.05')
2933
2934 physical_parameters_for_object_id_tray = {
2935     'sliding-friction': 0.1,
2936     'armature': 0.1,
2937     'stiffness': 0.0,
2938     'mass': 0.5,
2939     'damping': 20
2940 }
2941 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
2942
2943 physical_parameters_for_object_id_1 = { # bottle
2944     'sliding-friction': 0.22, # slight alignment adjustment
2945     'armature': 0.42, # fine-tune armature
2946     'stiffness': 0.54, # fine-tune stiffness
2947     'mass': 20.0, # mass remains unchanged
2948     'damping': 7.2 # slight fine-tune
2949 }
2950
2951 physical_parameters_for_object_id_2 = { # martini_glass
2952     'sliding-friction': 0.52, # slight increase
2953     'armature': 0.63, # slight adjustment
2954     'stiffness': 1.22, # slight fine-tune
2955     'mass': 10.0, # mass remains unchanged
2956     'damping': 8.1 # fine-tune damping
2957 }
2958
2959 physical_parameters_for_object_id_3 = { # wine_glass
2960     'sliding-friction': 0.73, # increase slightly
2961     'armature': 1.03, # slight adjustment
2962     'stiffness': 0.95, # fine-tune stiffness
2963     'mass': 4.0, # mass remains unchanged
2964     'damping': 8.0 # fine-tune damping
2965 }
2966
2967 sim.create_object(object_id=1, object_name='bottle', object_location=('
2968     row_1', 'column_3'), object_color='orange', object_physics=
2969     physical_parameters_for_object_id_1)
2970 sim.create_object(object_id=2, object_name='martini_glass',
2971     object_location=('row_1', 'column_2'), object_color='orange',
2972     object_physics=physical_parameters_for_object_id_2)
2973 sim.create_object(object_id=3, object_name='wine_glass', object_location
2974     =('row_1', 'column_1'), object_color='orange', object_physics=
2975     physical_parameters_for_object_id_3)
2976
2977 sim.create_scene()
2978 sim_out = sim.run_simulation()
2979 del sim
2980 ```

```

```

2970
2971 These refined adjustments should help get closer to the trajectory
2972 provided in `problem_trajectories.txt`:
2973
2974 - Slightly increased `sliding-friction` and `damping` for all objects to
2975 match their interaction with the tray more closely.
2976 - Fine-tuned `armature` and `stiffness` to slightly alter responses
2977 without overdoing adjustments.
2978
2979 Please test this configuration in your simulation environment to check
2980 for reduced trajectory prediction errors. Fine-tuning these
2981 configurations iteratively based on errors should help achieve the
2982 desired result.
2983
2984 =====
2985
2986 Thank you for providing the detailed analysis and results. Based on the
2987 previous attempts, it appears that the adjustments to the sliding-
2988 friction, armature, stiffness, and damping parameters need further
2989 fine-tuning to achieve a closer match with the trajectories in `
2990 problem_trajectories.txt`. I will make slight adjustments to these
2991 parameters to try to minimize the trajectory prediction error further
2992 .
2993
2994 **Adjusted Physical Parameters:**
2995
2996 ```python
2997 # refined_code.py
2998
2999 sim = SIMULATOR_MODEL()
3000 sim.create_pusher('3.0 3.0 0.05')
3001
3002 physical_parameters_for_object_id_tray = {
3003     'sliding-friction': 0.1,
3004     'armature': 0.1,
3005     'stiffness': 0.0,
3006     'mass': 0.5,
3007     'damping': 20
3008 }
3009 sim.create_tray(object_physics=physical_parameters_for_object_id_tray)
3010
3011 physical_parameters_for_object_id_1 = { # bottle
3012     'sliding-friction': 0.22, # slight adjustment for better alignment
3013     'armature': 0.41, # fine-tuning armature
3014     'stiffness': 0.54, # fine-tuning stiffness
3015     'mass': 20.0, # mass remains unchanged
3016     'damping': 7.3 # fine-tuning damping
3017 }
3018
3019 physical_parameters_for_object_id_2 = { # martini_glass
3020     'sliding-friction': 0.52, # fine-tuning sliding-friction
3021     'armature': 0.63, # fine-tuning armature
3022     'stiffness': 1.23, # fine-tuning stiffness
3023     'mass': 10.0, # mass remains unchanged
3024     'damping': 8.0 # fine-tuning damping
3025 }
3026
3027 physical_parameters_for_object_id_3 = { # wine_glass
3028     'sliding-friction': 0.72, # fine-tuning sliding-friction
3029     'armature': 1.04, # slight adjustment
3030     'stiffness': 0.94, # fine-tuning stiffness
3031     'mass': 4.0, # mass remains unchanged
3032     'damping': 8.0 # slight fine-tuning damping
3033 }

```



```

3024 sim.create_object(object_id=1, object_name='bottle', object_location=('
3025     row_1', 'column_3'), object_color='orange', object_physics=
3026     physical_parameters_for_object_id_1)
3027 sim.create_object(object_id=2, object_name='martini_glass',
3028     object_location=('row_1', 'column_2'), object_color='orange',
3029     object_physics=physical_parameters_for_object_id_2)
3030 sim.create_object(object_id=3, object_name='wine_glass', object_location
3031     =('row_1', 'column_1'), object_color='orange', object_physics=
3032     physical_parameters_for_object_id_3)
3033
3034 sim.create_scene()
3035 sim_out = sim.run_simulation()
3036 del sim
3037 ```
3038
3039 This refined code makes slight adjustments to the physical parameters,
3040 considering the previous results. The aim is to better align the
3041 simulated trajectories with the problem trajectories given in '
3042 problem_trajectories.txt'.
3043
3044 Let's run these refined parameters through the simulator and compare the
3045 generated trajectories with the reference ones.
3046
3047 Continue this iterative process, making small adjustments within the
3048 significant digits, to minimize the error further.
3049
3050 =====
3051
3052 Let's iteratively fine-tune the physical parameters for each object using
3053 the minimal changes approach. As each previous attempt had varied
3054 errors, I'll aim to adjust the parameters finely to achieve a closer
3055 alignment with the target trajectories.
3056
3057 Here is a refined attempt following the same code structure:
3058
3059 ```python
3060 # Generating refined code based on the provided feedback and previous
3061 attempts
3062
3063 # Initializing the simulator
3064 sim = SIMULATOR_MODEL()
3065 sim.create_pusher('3.0 3.0 0.05')
3066
3067 # Setting up the tray with the same parameters
3068 physical_parameters_for_object_id_tray = {
3069     'sliding-friction': 0.1,
3070     'armature': 0.1,
3071     'stiffness': 0.0,
3072     'mass': 0.5,
3073     'damping': 20
3074 }
3075 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3076
3077 # Setting up physical parameters for each object
3078 physical_parameters_for_object_id_1 = { # bottle, trying finer
3079     adjustments
3080     'sliding-friction': 0.22,
3081     'armature': 0.41,
3082     'stiffness': 0.54,
3083     'mass': 20.0,
3084     'damping': 7.2
3085 }
3086
3087 physical_parameters_for_object_id_2 = { # martini_glass, slight
3088     adjustments

```

```

3078     'sliding-friction': 0.53,
3079     'armature': 0.63,
3080     'stiffness': 1.23,
3081     'mass': 10.0,
3082     'damping': 8.0
3083 }
3084 physical_parameters_for_object_id_3 = { # wine_glass, finer tuning
3085     'sliding-friction': 0.73,
3086     'armature': 1.04,
3087     'stiffness': 0.95,
3088     'mass': 4.0,
3089     'damping': 7.9
3090 }
3091 # Creating objects on the tray
3092 sim.create_object(object_id=1, object_name='bottle', object_location=(
3093     row_1', 'column_3'), object_color='orange', object_physics=
3094     physical_parameters_for_object_id_1)
3095 sim.create_object(object_id=2, object_name='martini_glass',
3096     object_location=('row_1', 'column_2'), object_color='orange',
3097     object_physics=physical_parameters_for_object_id_2)
3098 sim.create_object(object_id=3, object_name='wine_glass', object_location
3099     =('row_1', 'column_1'), object_color='orange', object_physics=
3100     physical_parameters_for_object_id_3)
3101 # Creating the scene and running the simulation
3102 sim.create_scene()
3103 sim_out = sim.run_simulation()
3104 del sim
3105 ```
3106
3107 The rationale behind these values is as follows:
3108 - For 'bottle', the parameters were slightly fine-tuned to achieve
3109     balanced damping and appropriate friction for better alignment.
3110 - For 'martini_glass', adjustments were made to sliding friction and
3111     damping to fine-tune the alignment.
3112 - For 'wine_glass', tuning involved slight adjustments to friction,
3113     stiffness, and armature to refine the trajectory.
3114
3115 Let's run the simulation with these parameters to evaluate the trajectory
3116 prediction errors.
3117
3118 L LLMPHY OPTIMIZATION AND INTERACTION TRACE (PHASE2)
3119
3120 Similar to above, we present an example optimization steps from a phase 2 LLMPHY optimization.
3121
3122 Sure, here's the code for creating the provided scene based on '
3123     task_image_top_view_1.png' and 'task_image_side_view_2.png'. The
3124     object classes and colors are arranged as follows, based on the
3125     interpretation of the images.
3126
3127 ### Arrangement of Objects on the 3x3 Grid:
3128 - (1,1): Yellow (Bottle)
3129 - (1,2): Cyan (Bottle)
3130 - (1,3): Orange (Bottle)
3131 - (2,1): Brown (Martini Glass)
3132 - (2,2): Blue (Martini Glass)
3133 - (2,3): Pink (Martini Glass)
3134 - (3,1): Purple (Wine Glass)
3135 - (3,2): None
3136 - (3,3): Green (Wine Glass)

```

```
3132 Here's the code to create the scene using the specified physical
3133 attributes:
3134
3135 ```python
3136 sim = SIMULATOR_MODEL()
3137 sim.create_pusher('3.0 3.0 0.05')
3138 physical_parameters_for_object_id_tray = {
3139     'sliding-friction': 0.1,
3139     'armature': 0.1,
3140     'stiffness': 0.0,
3141     'mass': 0.5,
3142     'damping': 20
3143 }
3144 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3145
3146 # Physical parameters for bottle
3147 physical_parameters_for_bottle = {
3148     'sliding-friction': 0.21,
3148     'armature': 0.41,
3149     'stiffness': 0.55,
3150     'mass': 20.0,
3151     'damping': 7.0
3152 }
3153
3154 # Physical parameters for martini glass
3155 physical_parameters_for_martini_glass = {
3156     'sliding-friction': 0.52,
3157     'armature': 0.63,
3158     'stiffness': 1.22,
3159     'mass': 10.0,
3160     'damping': 8.0
3161 }
3162
3163 # Physical parameters for wine glass
3164 physical_parameters_for_wine_glass = {
3165     'sliding-friction': 0.72,
3166     'armature': 1.02,
3167     'stiffness': 0.94,
3168     'mass': 4.0,
3169     'damping': 7.9
3170 }
3171
3172 sim.create_object(object_id=1, object_name='bottle', object_location=('
3173 row_1', 'column_1'), object_color='yellow', object_physics=
3174 physical_parameters_for_bottle)
3175 sim.create_object(object_id=2, object_name='bottle', object_location=('
3176 row_1', 'column_2'), object_color='cyan', object_physics=
3177 physical_parameters_for_bottle)
3178 sim.create_object(object_id=3, object_name='bottle', object_location=('
3179 row_1', 'column_3'), object_color='orange', object_physics=
3180 physical_parameters_for_bottle)
3181 sim.create_object(object_id=4, object_name='martini_glass',
3182 object_location=('row_2', 'column_1'), object_color='brown',
3183 object_physics=physical_parameters_for_martini_glass)
3184 sim.create_object(object_id=5, object_name='martini_glass',
3185 object_location=('row_2', 'column_2'), object_color='blue',
3186 object_physics=physical_parameters_for_martini_glass)
3187 sim.create_object(object_id=6, object_name='martini_glass',
3188 object_location=('row_2', 'column_3'), object_color='pink',
3189 object_physics=physical_parameters_for_martini_glass)
3190 sim.create_object(object_id=7, object_name='wine_glass', object_location
3191 =('row_3', 'column_1'), object_color='purple', object_physics=
3192 physical_parameters_for_wine_glass)
3193
```

```

3186 sim.create_object(object_id=8, object_name='wine_glass', object_location
3187 =('row_3', 'column_3'), object_color='green', object_physics=
3188 physical_parameters_for_wine_glass)
3189
3190 sim.create_scene()
3191 sim_out=sim.run_simulation()
3192 del sim
3193 ```
3194 This code will recreate the scene as depicted in the provided images '
3195 task_image_top_view_1.png' and 'task_image_side_view_2.png' using the
3196 specific physical parameters given.
3197
3198 =====
3199 Certainly! Based on the feedback from the previous attempt and a careful
3200 review of the top-down view 'task_image_top_view_1.png', here is a
3201 refined version of the code to correctly match the object positions
3202 and colors.
3203
3204 ```python
3205 sim = SIMULATOR_MODEL()
3206 sim.create_pusher('3.0 3.0 0.05')
3207 physical_parameters_for_object_id_tray = {
3208     'sliding-friction': 0.1,
3209     'armature': 0.1,
3210     'stiffness': 0.0,
3211     'mass': 0.5,
3212     'damping': 20
3213 }
3214 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3215
3216 # Physical parameters for bottle
3217 physical_parameters_for_bottle = {
3218     'sliding-friction': 0.21,
3219     'armature': 0.41,
3220     'stiffness': 0.55,
3221     'mass': 20.0,
3222     'damping': 7.0
3223 }
3224
3225 # Physical parameters for martini glass
3226 physical_parameters_for_martini_glass = {
3227     'sliding-friction': 0.52,
3228     'armature': 0.63,
3229     'stiffness': 1.22,
3230     'mass': 10.0,
3231     'damping': 8.0
3232 }
3233
3234 # Physical parameters for wine glass
3235 physical_parameters_for_wine_glass = {
3236     'sliding-friction': 0.72,
3237     'armature': 1.02,
3238     'stiffness': 0.94,
3239     'mass': 4.0,
3240     'damping': 7.9
3241 }
3242
3243 # Object placements based on correct interpretation
3244 sim.create_object(object_id=1, object_name='bottle', object_location=('
3245 row_1', 'column_1'), object_color='yellow', object_physics=
3246 physical_parameters_for_bottle)
3247
3248

```

```

3240 sim.create_object(object_id=2, object_name='bottle', object_location=('
3241 row_1', 'column_2'), object_color='cyan', object_physics=
3242 physical_parameters_for_bottle)
3243 sim.create_object(object_id=3, object_name='bottle', object_location=('
3244 row_1', 'column_3'), object_color='orange', object_physics=
3245 physical_parameters_for_bottle)
3246 sim.create_object(object_id=4, object_name='martini_glass',
3247 object_location=('row_2', 'column_1'), object_color='brown',
3248 object_physics=physical_parameters_for_martini_glass)
3249 sim.create_object(object_id=5, object_name='wine_glass', object_location
3250 =('row_2', 'column_2'), object_color='blue', object_physics=
3251 physical_parameters_for_martini_glass)
3252 sim.create_object(object_id=6, object_name='martini_glass',
3253 object_location=('row_2', 'column_3'), object_color='pink',
3254 object_physics=physical_parameters_for_martini_glass)
3255 sim.create_object(object_id=7, object_name='wine_glass', object_location
3256 =('row_3', 'column_1'), object_color='purple', object_physics=
3257 physical_parameters_for_wine_glass)
3258 sim.create_object(object_id=8, object_name='wine_glass', object_location
3259 =('row_3', 'column_2'), object_color='brown', object_physics=
3260 physical_parameters_for_wine_glass)
3261
3262 sim.create_scene()
3263 sim_out=sim.run_simulation()
3264 del sim
3265 ```
3266
3267 Steps taken:
3268 - Identified that the issue with the object colors in previous Attempt
3269 #0.
3270 - 'cyan' should be 'blue'.
3271 - 'orange' should be 'purple'.
3272 - 'pink' should be 'brown'.
3273 - 'blue' should be 'cyan'.
3274 - Corrected these placements and colors to match the top-down view `
3275 task_image_top_view_1.png`.
3276
3277 Please run this corrected code in your simulator to check if it matches
3278 the desired output.
3279
3280 =====
3281
3282 Certainly! To reproduce the task images correctly, we need to ensure that
3283 each object is placed in the correct grid location and given the
3284 correct color. Let's carefully inspect the top-down image and the
3285 side-view image again:
3286
3287 From `task_image_top_view_1.png`:
3288 1. Top-left (row 1, column 1): Yellow object
3289 2. Top-middle (row 1, column 2): Cyan object
3290 3. Top-right (row 1, column 3): Orange object
3291 4. Middle-left (row 2, column 1): Brown object
3292 5. Middle-middle (row 2, column 2): Blue object
3293 6. Middle-right (row 2, column 3): Pink object
3294 7. Bottom-left (row 3, column 1): Purple object
3295 8. Bottom-middle (row 3, column 2): No object
3296 9. Bottom-right (row 3, column 3): Green object
3297
3298 From `task_image_side_view_2.png`, identify the types of objects:
3299 1. Bottles have a slender, tall shape.
3300 2. Martini glasses have a distinctive cone shape.
3301 3. Wine glasses have a bowl-like shape and are shorter than bottles.

```

```

3294
3295 Using these identifications and physical attributes provided, we will
3296     correct the mistake identified in the attempt. Below is the refined
3297     version of the Python code to correctly reproduce the top-down and
3298     side-view images of the scene.
3299
3300 ```python
3301 sim = SIMULATOR_MODEL()
3302 sim.create_pusher('3.0 3.0 0.05')
3303
3304 physical_parameters_for_object_id_tray = {
3305     'sliding-friction': 0.1,
3306     'armature': 0.1,
3307     'stiffness': 0.0,
3308     'mass': 0.5,
3309     'damping': 20
3310 }
3311 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3312
3313 # Physical parameters for bottle
3314 physical_parameters_for_bottle = {
3315     'sliding-friction': 0.21,
3316     'armature': 0.41,
3317     'stiffness': 0.55,
3318     'mass': 20.0,
3319     'damping': 7.0
3320 }
3321
3322 # Physical parameters for martini glass
3323 physical_parameters_for_martini_glass = {
3324     'sliding-friction': 0.52,
3325     'armature': 0.63,
3326     'stiffness': 1.22,
3327     'mass': 10.0,
3328     'damping': 8.0
3329 }
3330
3331 # Physical parameters for wine glass
3332 physical_parameters_for_wine_glass = {
3333     'sliding-friction': 0.72,
3334     'armature': 1.02,
3335     'stiffness': 0.94,
3336     'mass': 4.0,
3337     'damping': 7.9
3338 }
3339
3340 # Correct object placements and colors based on the images
3341 sim.create_object(object_id=1, object_name='bottle', object_location=('
3342     row_1', 'column_1'), object_color='yellow', object_physics=
3343     physical_parameters_for_bottle)
3344 sim.create_object(object_id=2, object_name='bottle', object_location=('
3345     row_1', 'column_2'), object_color='cyan', object_physics=
3346     physical_parameters_for_bottle)
3347 sim.create_object(object_id=3, object_name='bottle', object_location=('
3348     row_1', 'column_3'), object_color='orange', object_physics=
3349     physical_parameters_for_bottle)
3350 sim.create_object(object_id=4, object_name='martini_glass',
3351     object_location=('row_2', 'column_1'), object_color='brown',
3352     object_physics=physical_parameters_for_martini_glass)
3353 sim.create_object(object_id=5, object_name='bottle', object_location=('
3354     row_2', 'column_2'), object_color='blue', object_physics=
3355     physical_parameters_for_bottle)
3356 sim.create_object(object_id=6, object_name='martini_glass',
3357     object_location=('row_2', 'column_3'), object_color='pink',
3358     object_physics=physical_parameters_for_martini_glass)

```

```

3348 sim.create_object(object_id=7, object_name='martini_glass',
3349                   object_location=('row_3', 'column_1'), object_color='purple',
3350                   object_physics=physical_parameters_for_martini_glass)
3351 sim.create_object(object_id=8, object_name='wine_glass', object_location
3352                   =('row_3', 'column_3'), object_color='green', object_physics=
3353                   physical_parameters_for_wine_glass)
3354
3354 sim.create_scene()
3355 sim_out=sim.run_simulation()
3356 del sim
3357 ```
3358
3358 This code ensures that each object is assigned the correct position and
3359 color as per the task images. The corrected physical parameters will
3360 also contribute to a more accurate simulation output.
3361
3362 =====
3363
3363 To correctly reproduce the scene shown in the "task_image_top_view_1.png"
3364 and "task_image_side_view_2.png" images, we need to ensure that the
3365 placement, object types, and colors are all accurately represented.
3366 From GPT Attempt #0, we've identified that there are issues with the
3367 colors of the objects placed in the scene.
3368
3368 Let's refine the code to align the objects' colors and positions
3369 accurately according to the provided images.
3370
3371 Here's the corrected code snippet:
3372
3372 ```python
3373 sim = SIMULATOR_MODEL()
3374 sim.create_pusher('3.0 3.0 0.05')
3375
3376 physical_parameters_for_object_id_tray = {
3377     'sliding-friction': 0.1,
3378     'armature': 0.1,
3379     'stiffness': 0.0,
3380     'mass': 0.5,
3381     'damping': 20
3382 }
3382 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3383
3384 # Physical parameters for bottle
3384 physical_parameters_for_bottle = {
3385     'sliding-friction': 0.21,
3386     'armature': 0.41,
3387     'stiffness': 0.55,
3388     'mass': 20.0,
3389     'damping': 7.0
3390 }
3391
3391 # Physical parameters for martini glass
3392 physical_parameters_for_martini_glass = {
3393     'sliding-friction': 0.52,
3394     'armature': 0.63,
3395     'stiffness': 1.22,
3396     'mass': 10.0,
3397     'damping': 8.0
3398 }
3398
3398 # Physical parameters for wine glass
3399 physical_parameters_for_wine_glass = {
3400     'sliding-friction': 0.72,
3401     'armature': 1.02,
3402     'stiffness': 0.94,

```

```

3402     'mass': 4.0,
3403     'damping': 7.9
3404 }
3405
3406 # Correct object placements and colors based on the images
3407 sim.create_object(object_id=1, object_name='bottle', object_location=('
3408     row_1', 'column_1'), object_color='yellow', object_physics=
3409     physical_parameters_for_bottle)
3410 sim.create_object(object_id=2, object_name='bottle', object_location=('
3411     row_1', 'column_2'), object_color='cyan', object_physics=
3412     physical_parameters_for_bottle)
3413 sim.create_object(object_id=3, object_name='bottle', object_location=('
3414     row_1', 'column_3'), object_color='orange', object_physics=
3415     physical_parameters_for_bottle)
3416 sim.create_object(object_id=4, object_name='martini_glass',
3417     object_location=('row_2', 'column_1'), object_color='brown',
3418     object_physics=physical_parameters_for_martini_glass)
3419 sim.create_object(object_id=5, object_name='bottle', object_location=('
3420     row_2', 'column_2'), object_color='blue', object_physics=
3421     physical_parameters_for_bottle)
3422 sim.create_object(object_id=6, object_name='wine_glass', object_location
3423     =('row_2', 'column_3'), object_color='pink', object_physics=
3424     physical_parameters_for_wine_glass) # Corrected to wine glass
3425 sim.create_object(object_id=7, object_name='martini_glass',
3426     object_location=('row_3', 'column_1'), object_color='purple',
3427     object_physics=physical_parameters_for_martini_glass) # Corrected
3428     color position
3429 sim.create_object(object_id=8, object_name='wine_glass', object_location
3430     =('row_3', 'column_3'), object_color='green', object_physics=
3431     physical_parameters_for_wine_glass)
3432
3433 sim.create_scene()
3434 sim_out=sim.run_simulation()
3435 del sim
3436 ```
3437
3438 Changes made:
3439 1. For 'object_id=6', the type was changed to 'wine_glass' instead of '
3440     martini_glass'.
3441 2. The color 'purple' was moved to position '(row_3, column_1)' for the '
3442     martini_glass' as per the task image.
3443
3444 Ensure the scene setup provided above now accurately reflects the task
3445     images. With these corrections, the positions and object types should
3446     match the top-down and side view images respectively.
3447
3448 =====
3449
3450 To refine the Python code for generating the scene to exactly match the
3451     given images ('task_image_top_view_1.png' and 'task_image_side_view_2
3452     .png'), we need to correct the object colors and their placements on
3453     the grid. Below is the mistake analysis from GPT Attempt #0 and the
3454     necessary corrections:
3455
3456 1. **Object at '(row_1, column_2)**': The given image shows this is
3457     yellow, but the existing code has it as cyan.
3458 2. **Object at '(row_1, column_3)**': The given image shows this is cyan,
3459     but the existing code has it as orange.
3460 3. **Object at '(row_3, column_1)**': The given image shows this is
3461     orange, but the existing code has it as purple.
3462
3463 We can use this information to adjust the code appropriately:
3464
3465 1. Place the cyan bottle at '(row_1, column_3)'.
3466 2. Place the orange bottle at '(row_3, column_1)'.

```



```

3456 3. Place the purple martini glass at `(row_2, column_2)`.
3457
3458 Let's write the corrected code accordingly:
3459
3460 ```python
3461 sim = SIMULATOR_MODEL()
3462 sim.create_pusher('3.0 3.0 0.05')
3463
3464 physical_parameters_for_object_id_tray = {
3465     'sliding-friction': 0.1,
3466     'armature': 0.1,
3467     'stiffness': 0.0,
3468     'mass': 0.5,
3469     'damping': 20
3470 }
3471 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3472
3473 # Physical parameters for bottle
3474 physical_parameters_for_bottle = {
3475     'sliding-friction': 0.21,
3476     'armature': 0.41,
3477     'stiffness': 0.55,
3478     'mass': 20.0,
3479     'damping': 7.0
3480 }
3481
3482 # Physical parameters for martini glass
3483 physical_parameters_for_martini_glass = {
3484     'sliding-friction': 0.52,
3485     'armature': 0.63,
3486     'stiffness': 1.22,
3487     'mass': 10.0,
3488     'damping': 8.0
3489 }
3490
3491 # Physical parameters for wine glass
3492 physical_parameters_for_wine_glass = {
3493     'sliding-friction': 0.72,
3494     'armature': 1.02,
3495     'stiffness': 0.94,
3496     'mass': 4.0,
3497     'damping': 7.9
3498 }
3499
3500 # Correct object placements and colors based on the images
3501 sim.create_object(object_id=1, object_name='bottle', object_location=(
3502     'row_1', 'column_1'), object_color='yellow', object_physics=
3503     physical_parameters_for_bottle)
3504 sim.create_object(object_id=2, object_name='bottle', object_location=(
3505     'row_1', 'column_3'), object_color='cyan', object_physics=
3506     physical_parameters_for_bottle) # Corrected to cyan and position
3507 sim.create_object(object_id=3, object_name='bottle', object_location=(
3508     'row_3', 'column_1'), object_color='orange', object_physics=
3509     physical_parameters_for_bottle) # Corrected to orange and position
3510 sim.create_object(object_id=4, object_name='martini_glass',
3511     object_location=('row_2', 'column_1'), object_color='brown',
3512     object_physics=physical_parameters_for_martini_glass)
3513 sim.create_object(object_id=5, object_name='bottle', object_location=(
3514     'row_2', 'column_2'), object_color='blue', object_physics=
3515     physical_parameters_for_bottle)
3516 sim.create_object(object_id=6, object_name='wine_glass', object_location
3517     =('row_2', 'column_3'), object_color='pink', object_physics=
3518     physical_parameters_for_wine_glass)
3519 sim.create_object(object_id=7, object_name='martini_glass',
3520     object_location=('row_1', 'column_2'), object_color='purple',

```

```

3510     object_physics=physical_parameters_for_martini_glass) # Corrected to
3511     purple and position
3512 sim.create_object(object_id=8, object_name='wine_glass', object_location
3513    =('row_3', 'column_3'), object_color='green', object_physics=
3514     physical_parameters_for_wine_glass)
3515
3516 sim.create_scene()
3517 sim_out=sim.run_simulation()
3518 del sim
3519 ```
3520
3521 This code should now correctly reproduce the task images:
3522
3523 - **Object Placements**::
3524 - `(row_1, column_1)` -> Yellow bottle
3525 - `(row_1, column_2)` -> Purple martini glass
3526 - `(row_1, column_3)` -> Cyan bottle
3527 - `(row_2, column_1)` -> Brown martini glass
3528 - `(row_2, column_2)` -> Blue bottle
3529 - `(row_2, column_3)` -> Pink wine glass
3530 - `(row_3, column_1)` -> Orange bottle
3531 - `(row_3, column_3)` -> Green wine glass
3532
3533 With these corrected attributes and proper placements, running the
3534 simulation should yield the desired images.
3535
3536 =====
3537
3538 Thank you for providing the various details and previous attempts. Let's
3539 analyze the array of object placements and the corresponding colors
3540 more carefully, given both views and the physical properties.
3541
3542 The given task_images (task_image_top_view_1.png and
3543 task_image_side_view_2.png) indicate a specific arrangement of
3544 colored objects. Let's identify and correct the colors and positions
3545 of the objects in the generated scene to match the task:
3546
3547 1. Yellow object: A bottle, located in row 1, column 1.
3548 2. Cyan object: A bottle, located in row 1, column 3.
3549 3. Orange object: A bottle, located in row 1, column 2.
3550 4. Purple object: A martini glass, located in row 3, column 2.
3551 5. Blue object: A bottle, located in row 2, column 2.
3552 6. Pink object: A wine glass, located in row 3, column 1.
3553 7. Green object: A wine glass, located in row 3, column 3.
3554 8. Brown object: A martini glass, located in row 2, column 1.
3555
3556 Using the corrected layout information, here is the refined Python code
3557 to reproduce the task images correctly:
3558
3559 ```python
3560 sim = SIMULATOR_MODEL()
3561 sim.create_pusher('3.0 3.0 0.05')
3562
3563 # Physical parameters for the tray
3564 physical_parameters_for_object_id_tray = {
3565     'sliding-friction': 0.1,
3566     'armature': 0.1,
3567     'stiffness': 0.0,
3568     'mass': 0.5,
3569     'damping': 20
3570 }
3571 sim.create_tray(object_physics = physical_parameters_for_object_id_tray)
3572
3573 # Physical parameters for bottle
3574 physical_parameters_for_bottle = {

```

```

3564     'sliding-friction': 0.21,
3565     'armature': 0.41,
3566     'stiffness': 0.55,
3567     'mass': 20.0,
3568     'damping': 7.0
3569 }
3570 # Physical parameters for martini glass
3571 physical_parameters_for_martini_glass = {
3572     'sliding-friction': 0.52,
3573     'armature': 0.63,
3574     'stiffness': 1.22,
3575     'mass': 10.0,
3576     'damping': 8.0
3577 }
3578 # Physical parameters for wine glass
3579 physical_parameters_for_wine_glass = {
3580     'sliding-friction': 0.72,
3581     'armature': 1.02,
3582     'stiffness': 0.94,
3583     'mass': 4.0,
3584     'damping': 7.9
3585 }
3586 # Correct object placements and colors based on the task images
3587 sim.create_object(object_id=1, object_name='bottle', object_location=('
3588 row_1', 'column_1'), object_color='yellow', object_physics=
3589 physical_parameters_for_bottle)
3590 sim.create_object(object_id=2, object_name='bottle', object_location=('
3591 row_1', 'column_3'), object_color='cyan', object_physics=
3592 physical_parameters_for_bottle)
3593 sim.create_object(object_id=3, object_name='bottle', object_location=('
3594 row_1', 'column_2'), object_color='orange', object_physics=
3595 physical_parameters_for_bottle)
3596 sim.create_object(object_id=4, object_name='martini_glass',
3597 object_location=('row_2', 'column_1'), object_color='brown',
3598 object_physics=physical_parameters_for_martini_glass)
3599 sim.create_object(object_id=5, object_name='bottle', object_location=('
3600 row_2', 'column_2'), object_color='blue', object_physics=
3601 physical_parameters_for_bottle)
3602 sim.create_object(object_id=6, object_name='wine_glass', object_location
3603 =('row_3', 'column_1'), object_color='pink', object_physics=
3604 physical_parameters_for_wine_glass)
3605 sim.create_object(object_id=7, object_name='wine_glass', object_location
3606 =('row_3', 'column_3'), object_color='green', object_physics=
3607 physical_parameters_for_wine_glass)
3608 sim.create_object(object_id=8, object_name='martini_glass',
3609 object_location=('row_3', 'column_2'), object_color='purple',
3610 object_physics=physical_parameters_for_martini_glass)
3611 sim.create_scene()
3612 sim_out=sim.run_simulation()
3613 del sim
3614 ```
3615
3616 This code corrects the placement and color assignments for the given
3617 objects, ensuring they match what is observed in the task image. Run
3618 this code in your simulator to verify the outputs.
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637

```