RoboPlayground: Democratizing Robotic Evaluation through Structured Physical Domains

Yi Ru Wang*[1,2], Carter Ung*[1], Evan Gubarev[1], Christopher Tan[1], Siddhartha Srinivasa†[1], Dieter Fox†[1,2]

*Equal contribution. †Equal advising.

[1]University of Washington, Seattle, WA, USA

[2]Allen Institute for Artificial Intelligence, Seattle, WA, USA

yiruwang@cs.washington.edu, carterung@gmail.com

Abstract

Evaluation of robotic manipulation systems has largely relied on fixed benchmarks authored by a small number of experts, where task instances, constraints, and success criteria are predefined and difficult to extend. This paradigm limits who can shape evaluation and obscures how policies respond to user-authored variations in task intent, constraints, and notions of success. We argue that evaluating modern manipulation policies requires reframing evaluation as a language-driven process over structured physical domains. We present RoboPlayground, a framework that enables users to author executable manipulation tasks using natural language within a structured physical domain. Natural language instructions are compiled into reproducible task specifications with explicit asset definitions, initialization distributions, and success predicates. Each instruction defines a structured family of related tasks, enabling controlled semantic and behavioral variation while preserving executability and comparability. We instantiate RoboPlayground in a structured block manipulation domain and evaluate it along three axes. A user study shows that the language-driven interface is easier to use and imposes lower cognitive workload than programming-based and code-assist baselines. Evaluating learned policies on language-defined task families reveals generalization failures that are not apparent under fixed benchmark evaluations. Finally, we show that task diversity scales with contributor diversity rather than task count alone, enabling evaluation spaces to grow continuously through crowd-authored contributions.

1. Introduction

Who gets to decide what it means for a robot to be competent? Today, robotic manipulation systems are evaluated almost exclusively through benchmarks designed by a small number of experts. These benchmarks specify fixed task instances, success conditions, and evaluation protocols, implicitly encoding which behaviors matter and which variations are worth testing. While this paradigm has driven substantial progress, it centralizes control over evaluation and limits both who can define evaluation tasks and what questions can be asked about a system's behavior.

Similar limitations have appeared in other domains. In visual reasoning, diagnostic datasets such as CLEVR reframed evaluation around controlled task generation within a structured domain [12]. By focusing on interpretable primitives rather than maximal realism, CLEVR shifted evaluation from static instances to structured families of tasks, enabling clearer attribution of failure modes. In contrast, work in natural language processing has emphasized the dynamics of evaluation over time. Dynamic benchmarking efforts such as Dynabench treat evaluation as a human-in-the-loop process, where users iteratively generate and refine examples to surface model weaknesses [14]. Together, these efforts highlight two complementary principles for informative evaluation: structured task spaces that make variation interpretable, and participatory mechanisms that allow evaluation to grow continuously beyond its initial design.

Evaluating modern manipulation policies requires embracing both principles. An effective evaluation system should satisfy four key desiderata. First, it must be **accessible**, allowing users to express task intent, constraints, and success criteria using natural language without expertise in simulation internals or benchmark-specific code. Second, it should support **continuous growth**, enabling the evaluation space to expand over time through contributions from many users rather than remaining fixed. Third, it must ensure **reproducibility**, so that tasks can be precisely re-executed across models and evaluations, enabling fair comparison as the evaluation space grows. Finally, it should provide **structured control**, constraining user-authored instructions to remain executable and interpretable while enabling systematic variation and meaningful attribution of failure modes.

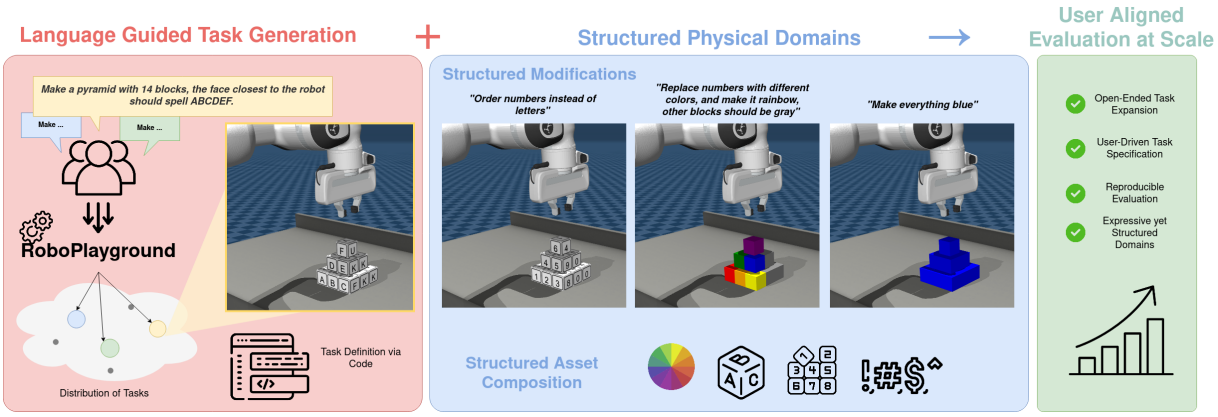


Figure 1. **Language-Guided Task Generation in Structured Physical Domains.** Natural language instructions are compiled into executable task templates within RoboPlayground, enabling open-ended task generation while preserving structure and control. Structured domains support systematic steering through controlled variations in task semantics, constraints, and asset composition, enabling expressive, reproducible, and extensible evaluation.

We propose reframing robotic manipulation evaluation as a language-driven, user-authored process over structured physical domains, shifting evaluation from static expert-defined benchmarks to an accessible, reproducible, and continuously expanding task space.

In this work, we present RoboPlayground, a language-driven framework for defining robotic manipulation tasks in a structured physical domain. Natural language serves as the primary authoring interface, allowing users to specify task intent, constraints, and success conditions without interacting with benchmark-specific code. Each instruction is compiled into an executable task specification with explicit definitions of assets, initialization distributions, and success predicates, enabling tasks to be precisely re-executed across models and evaluations. Rather than yielding a single fixed task instance, each instruction defines a family of related tasks, providing a continuously growing evaluation space over time while preserving controlled, comparable structure.

In summary, this paper makes three contributions. (1) We introduce a language-driven evaluation framework that democratizes manipulation evaluation by making task specification accessible to non-expert users while maintaining reproducibility. (2) We empirically demonstrate that language-defined task families reveal policy behaviors and limitations that are missed by conventional instance-based benchmarks. (3) We show that evaluation spaces can grow continuously through user- and model-authored instructions without sacrificing comparability or scientific rigor.

2. Related Works

2.1. Evaluation and Benchmarking for Robotic Manipulation

Robotic manipulation systems are most commonly evaluated using fixed benchmark suites composed of predefined task instances, environments, and success criteria, such as RLBench [11], LIBERO [20], RoboCasa [23], Behaviour-1k [16], ManiSkill [22], Colosseum [26], Simpler [17], and RoboEval [30]. While these benchmarks have been instrumental in standardizing evaluation, they define evaluation over a fixed and finite set of expert-authored tasks, with task structure, constraints, and success criteria encoded procedurally and not exposed for user modification. Although some benchmarks include natural language annotations or language-conditioned tasks, language is typically treated as documentation or policy input rather than as part of the executable task specification, making it difficult to introduce task variations or alternative notions of success without modifying benchmark code. Recent work has explored complementary directions for scaling evaluation: RoboArena [3] democratizes who evaluates and where evaluation occurs through crowd-sourced, double-blind pairwise comparisons over unconstrained real-world tasks, while Polaris [10] improves the fidelity and scalability of simulation-based evaluation via real-to-sim scene reconstruction, but retains fixed, expert-authored tasks and success criteria. In contrast, our work focuses on democratizing what is evaluated by enabling users to author, modify, and refine executable task specifications through language, while preserving structure, reproducibility, and comparability.

2.2. LLM-Based Task and Environment Generation

Recent work has explored using large language models to generate robotic tasks, environments, rewards, or curricula at scale. Systems such as GenSim [28], Gen2Sim [13], RoboGen [29], and AnyTask [7] leverage LLMs to synthesize tasks or simulation assets, while Eureka [21] and Eureka-verse [19] use language models to automatically generate reward functions or learning curricula. These approaches primarily target data generation and training diversity, rather than evaluation itself: generated tasks are treated as inputs to learning pipelines, and the task space is not exposed to

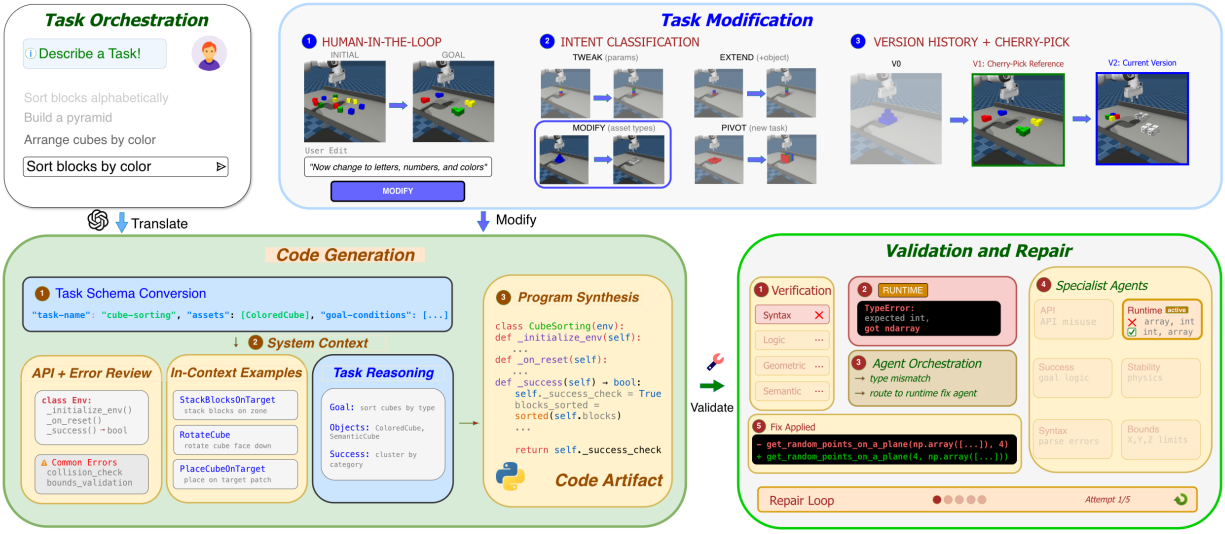


Figure 2. **System overview.** (*Left*) A user provides a natural language task description, which is compiled into a structured schema specifying assets, initialization logic, and success conditions. The system synthesizes an executable implementation conditioned on this schema and retrieved context (APIs, prior tasks, error patterns). (*Bottom right*) Tasks are admitted only after passing a multi-stage validation and repair pipeline enforcing software correctness, API consistency, and physical realizability. (*Top right*) Validated artifacts support context-aware steering: modification requests are classified into intent categories (*tweak*, *extend*, *modify*, *pivot*), with all variants tracked via explicit version history to preserve lineage and comparability.

users as a controllable or interpretable evaluation interface. Task generation is typically decoupled from mechanisms for enforcing reproducibility, tracking task lineage, or systematically relating task variations to evaluation outcomes. Language has also been used to guide robot behavior at execution time, as in SayCan [1] and Code as Policies [18], where it serves as a high-level planning or control signal while task definitions and success criteria remain fixed and externally specified. In contrast, our work treats language as a first-class interface for evaluation: natural language instructions are compiled into structured, executable task specifications with explicit asset definitions, initialization distributions, and success predicates, enabling users to author reproducible families of semantically related evaluation tasks that support controlled variation and systematic comparison.

3. Methods

This section describes how the framework operationalizes the core desiderata: accessibility, continuous growth, reproducibility, and structured control. We first define the design principles and formalize the task representation in Section 3.1. We then describe the task orchestration process that make language-defined manipulation tasks executable and interpretable in Section 3.2. We then describe how natural language instructions are compiled into concrete, reproducible task artifacts through a validation pipeline that enforces physical realizability and consistency Section 3.3. Finally, we introduce a context-aware steering mechanism that enables users to systematically vary tasks and expand the evaluation space over time while preserving explicit comparability between task variants in Section 3.4.

3.1. Task Representation and Design Principles

Our framework treats natural language as an executable interface for task specification. User instructions are compiled into concrete task realizations that fully determine assets, initialization logic, and success conditions. Rather than producing isolated benchmark instances, the system yields reusable task artifacts that can be shared, re-executed, and systematically varied, enabling evaluation spaces to grow through user contribution without sacrificing scientific rigor. Figure 2 provides an overview of this process. Natural language instructions are compiled into structured task proposals, synthesized into executable implementations, and admitted as task artifacts only after passing a multi-stage validation pipeline. Validated artifacts can then be iteratively refined through context-aware steering, enabling controlled task variation while preserving reproducibility and explicit lineage.

Formally, we define a manipulation task as a tuple

$$T = (\mathcal{A}, \rho_0, G, \ell, \mathcal{V}), \tag{1}$$

where $\mathcal{A}$ denotes the set of task assets, ρ_0 is a distribution over initial states, $G : \mathcal{S} \to \{0, 1\}$ is a success predicate over simulator states, ℓ is the canonical natural language instruction, and $\mathcal{V}$ is a set of paraphrases used for robustness testing.

This decomposition reflects a deliberate design choice. Logical equivalence at the level of language does not imply equivalence of task realization: differences in tolerances, reset distribution, or success-check timing can lead to divergent evaluation outcomes even when tasks

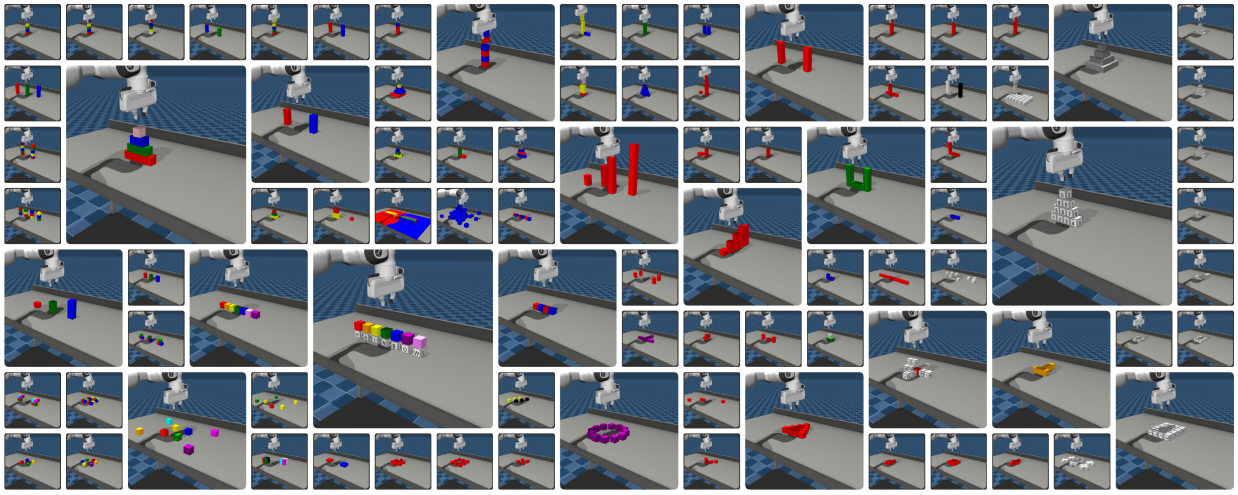


Figure 3. **Sample of Language-Defined Manipulation Tasks.** Executable tasks spanning geometric constructions, spatial alignment, and semantically constrained arrangements, organized by proximity in a learned task-embedding space. Coherent clusters correspond to families with shared attributes, spatial relations, and success definitions (e.g., stacking, ordering, grouping, alignment).

are described identically. Language alone is therefore insufficient as a unit of evaluation.

3.2. Task Orchestration Through Language

Given a natural language description u, task construction begins by translating language into a structured representation of task intent. Specifically, the system infers and populates a fixed `TaskSchema` that explicitly specifies the task name, relevant assets, goal conditions, and initialization logic. The use of a fixed schema ensures that all task-relevant fields are present and disambiguated before execution, enabling complete interpretation of the instruction and preventing underspecified task definitions.

Conditioned on the validated task schema, the system then synthesizes an executable task implementation. This process leverages an LLM with access to relevant environment APIs, prior task implementations, and diagnostic error information retrieved based on structural similarity to the proposed task. The LLM produces an intermediate natural-language task specification that articulates the intended objects, goal configuration, and success criteria, which is subsequently compiled into executable code.

Executable tasks are implemented as classes that extend a fixed environment interface. Each task defines methods for environment initialization, reset-time sampling from ρ_0, and success evaluation corresponding to G. This constrained interface enforces uniform structure across task implementations and limits variation arising from authoring style, ensuring that differences in evaluation outcomes reflect task content rather than implementation artifacts.

3.3. Validation and Physical Realizability

Language-defined tasks are only meaningful if they are both executable and physically realizable. Each synthesized task implementation is therefore subjected to a multi-stage validation pipeline before being admitted as a task artifact.

Basic validation. Basic validation enforces software correctness independent of physics simulation. Generated code is subjected to static analysis to detect syntactic errors and forbidden patterns, compiled in an isolated execution environment to detect import and definition errors, and instantiated to detect runtime failures during object creation or reset-time sampling.

Goal-state verification. To enforce physical realizability of the success predicate, tasks are instantiated in the goal configuration and simulated forward under zero action to allow contacts to settle. The success predicate G must evaluate to true after settling and remain true over an extended horizon, ensuring that the goal configuration is both achievable and stable under the simulator's physics model.

Iterative repair. When validation fails, the failure is classified according to its source (e.g., syntax, API usage, runtime instantiation, goal satisfaction, or physical instability), and a corresponding repair operator proposes a localized modification to the task implementation. Repairs may adjust object placements, relax geometric constraints, or rewrite components of the success predicate, depending on the failure type. Validation is then re-run on the repaired implementation. This process repeats until all checks pass or a fixed retry budget is exhausted, ensuring that admitted tasks satisfy executability and physical consistency while remaining faithful to the original language intent.

3.4. Controlled Task Modifications

A validated task artifact defines a reference task instance from which a family of related tasks can be derived. To enable systematic task variation without sacrificing comparability, the framework provides a context-aware

Table 1. User study results (N=26). Mean ± 95% CI margin. SUS: 0–100 scale. TLX: unweighted mean of five NASA-TLX subscales, normalized to 0–100. Rank: 1=best. Preferred: share selecting each system overall.

System	SUS ↑	TLX workload ↓	Usability rank ↓	Preferred (%)
GenSim	52.5±9.3	41.8±9.0	2.7±0.2	8
Cursor	68.8±7.8	36.7±10.4	2.0±0.2	23
RoboPlayground	**83.4±6.9**	**18.6±7.7**	**1.3±0.3**	**69**

steering mechanism that interprets user modification requests and constrains how tasks may evolve.

Given a modification request, the system first interprets the intent and extracts structured parameters such as dimensional changes, ordering constraints, or asset-type substitutions. It then classifies the request into one of five steering categories: *Tweak*, *Extend*, *Modify*, *Pivot*, or *Fresh*. Each category specifies explicit preservation guarantees over the task components $(\mathcal{A}, \rho_0, G)$. For example, *Tweak* and *Extend* preserve the original task structure and success predicate, enabling direct comparability with the reference task, while *Modify* and *Pivot* permit progressively broader semantic or structural changes when required by the user intent.

Task evolution is tracked through versioned snapshots that record structured summaries of assets, goals, and code hashes. When a modification requires asset types incompatible with the current version, the system selects a compatible prior snapshot as the reference. This allows coherent multi-step refinement without manual bookkeeping. Each validated variant produces a new snapshot, yielding version-controlled task families with explicit lineage suitable for systematic evaluation and controlled analysis of task variation.

4. Results

We evaluate ROBOPLAYGROUND along three axes that are central to its role as a democratized evaluation framework for robotic manipulation: (i) the usability of its task authoring interface, (ii) the diagnostic value of the resulting task set for assessing policy generalization, and (iii) the scalability of task creation under open-world, crowd-driven use. Across all experiments, we focus on whether ROBOPLAYGROUND enables task specifications that are both easier to author and more informative for evaluation than existing alternatives.

4.1. Usability of the Task Authoring Interface

Experimental setting. We evaluate the usability of RoboPlayground in comparison to two baseline task authoring interfaces, GenSim [28] and Cursor [2], using a within-subjects user study ($N = 26$). Participants were asked to construct an identical manipulation task (build a 3D structure using blocks under various constraints) using each system. All participants interacted with all three systems, enabling paired comparisons of perceived usability, cognitive workload, and user preference. We measure usability using the System Usability Scale (SUS) [5], cognitive workload using NASA-TLX subscales [8], and overall preference through usability and forced-choice rankings. Results are summarized in Table 1.

RoboPlayground achieves higher perceived usability than baselines. Across participants (N=26), RoboPlayground attains the highest System Usability Scale (SUS) score (83.4 ± 6.9; mean ± 95% confidence interval margin), well above the conventional acceptability threshold of 68. GenSim and Cursor achieve substantially lower mean SUS (52.5 ± 9.3 and 68.8 ± 7.8, respectively). Paired Wilcoxon signed-rank tests confirm that RoboPlayground significantly outperforms both GenSim (p<0.001) and Cursor (p=0.0017), so the advantage is not limited to the weakest baseline: RoboPlayground is rated more usable than a strong general-purpose assistant interface as well. The interval for GenSim is the widest of the three, consistent with more heterogeneous experiences in that condition, whereas RoboPlayground shows the tightest margin among systems, indicating comparatively consistent high ratings.

RoboPlayground reduces perceived cognitive workload relative to baselines. Cognitive workload is summarized as the unweighted mean of five NASA-TLX subscales (Mental Demand, Temporal Demand, Effort, Frustration, and reversed Performance), each normalized to 0–100 and oriented so that lower is better. RoboPlayground yields the lowest mean composite score (18.6 ± 7.7; mean ± 95% confidence interval margin), compared to 41.8 ± 9.0 for GenSim and 36.7 ± 10.4 for Cursor. Paired Wilcoxon signed-rank tests show that RoboPlayground significantly reduces perceived workload relative to both GenSim (p=0.0007) and Cursor (p=0.0019). GenSim and Cursor do not differ significantly from each other on this composite (p=0.22), whereas RoboPlayground separates clearly from each baseline; Cursor also exhibits the widest TLX margin among the three, indicating somewhat more spread in workload ratings even though the paired comparison to RoboPlayground remains significant.

Participants consistently prefer RoboPlayground over baseline interfaces. Subjective measures reinforce the quantitative usability and workload results. RoboPlayground achieves the best mean usability rank (1.3 ± 0.3; lower is better), with GenSim and Cursor at 2.7 ± 0.2 and 2.0 ± 0.2, respectively. A Friedman test shows strong differences in rankings across systems (p<0.001), and post-hoc paired Wilcoxon tests confirm that RoboPlayground is ranked significantly better than both GenSim (p=0.0001) and Cursor (p=0.0078). In forced-choice overall preference, 69% of participants select RoboPlayground, compared to 23% for Cursor and 8% for GenSim. A chi-square goodness-of-fit test rejects a uniform split across the three options (p=0.0003), consistent with concentration of preference on RoboPlayground. Together, the ranking and preference distributions indicate a stable, statistically

Table 2. **Results on in-distribution and generalization tasks.** Success rates (%) ± standard errors across six policies on training tasks (top) and held-out generalization tasks (bottom); best per task in **bold**. Generalization tasks perturb training tasks along semantic (S), visual (V), and behavioural (B) axes per [6].

Method	Red Block Front of Yellow	Red Block Right Placement	Red Behind Yellow	Red Block Stacking	Three Block Color Stacking	Red Block Left Placement	Red on Yellow Stack	Yellow on Red Stack	Color Block Alignment	Place Two Blocks on Patch
Pi-0.5	72.0 ± 6.3	68.0 ± 6.6	58.0 ± 7.0	0.0 ± 0.0	6.0 ± 3.4	64.0 ± 6.8	46.0 ± 7.0	62.0 ± 6.9	0.0 ± 0.0	74.0 ± 6.2
Pi-0.5 (LoRA)	32.0 ± 6.6	16.0 ± 5.2	48.0 ± 7.1	0.0 ± 0.0	0.0 ± 0.0	12.0 ± 4.6	12.0 ± 4.6	36.0 ± 6.8	0.0 ± 0.0	28.0 ± 6.3
Adapter	66.0 ± 6.7	64.0 ± 6.8	60.0 ± 6.9	0.0 ± 0.0	2.0 ± 2.0	26.0 ± 6.2	56.0 ± 7.0	54.0 ± 7.0	2.0 ± 2.0	78.0 ± 5.9
Dual	**88.0 ± 4.6**	76.0 ± 6.0	**74.0 ± 6.2**	2.0 ± 2.0	12.0 ± 4.6	**84.0 ± 5.2**	64.0 ± 6.8	54.0 ± 7.0	6.0 ± 3.4	84.0 ± 5.2
GR00T	82.0 ± 5.4	84.0 ± 5.2	**74.0 ± 6.2**	**10.0 ± 4.2**	**22.0 ± 5.9**	68.0 ± 6.6	66.0 ± 6.7	56.0 ± 7.0	**12.0 ± 4.6**	**96.0 ± 2.8**
Qwen-OFT	82.0 ± 5.4	**86.0 ± 4.9**	68.0 ± 6.6	2.0 ± 2.0	14.0 ± 4.9	**84.0 ± 5.2**	**76.0 ± 6.0**	**68.0 ± 6.6**	2.0 ± 2.0	78.0 ± 5.9

Perturbation	S+B	V	S	S+V	V+B	S	V	S+B	V	V	S+B	V
Method	Red Block Two Tower	Blue Block Stacking	Yellow Block Left Placement	Red Block Left of Blue	Yellow on Red Unstack Restack	Green on Blue Stack	Three Block Perturbed	Three Block Beside	Place Two Blue Blocks on Patch	Place Two Blocks on Green Patch	Stack Two Blocks on Patch	Place Two Blocks on Long Patch
Pi-0.5	4.0 ± 2.8	0.0 ± 0.0	76.0 ± 6.0	50.0 ± 7.1	0.0 ± 0.0	60.0 ± 6.9	10.0 ± 4.2	12.0 ± 4.6	46.0 ± 7.0	80.0 ± 5.7	0.0 ± 0.0	**68.0 ± 6.6**
Pi-0.5 (LoRA)	2.0 ± 2.0	0.0 ± 0.0	10.0 ± 4.2	32.0 ± 6.6	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 2.0	0.0 ± 0.0	10.0 ± 4.2	30.0 ± 6.5	**2.0 ± 2.0**	28.0 ± 6.3
Adapter	0.0 ± 0.0	0.0 ± 0.0	4.0 ± 2.8	36.0 ± 6.8	0.0 ± 0.0	0.0 ± 0.0	8.0 ± 3.8	0.0 ± 0.0	46.0 ± 7.0	38.0 ± 6.9	0.0 ± 0.0	26.0 ± 6.2
Dual	8.0 ± 3.8	0.0 ± 0.0	74.0 ± 6.2	**60.0 ± 6.9**	0.0 ± 0.0	56.0 ± 7.0	8.0 ± 3.8	10.0 ± 4.2	80.0 ± 5.7	72.0 ± 6.3	**2.0 ± 2.0**	54.0 ± 7.0
GR00T	**10.0 ± 4.2**	0.0 ± 0.0	**78.0 ± 5.9**	52.0 ± 7.1	**2.0 ± 2.0**	62.0 ± 6.9	**20.0 ± 5.7**	4.0 ± 2.8	**86.0 ± 4.9**	**90.0 ± 4.2**	0.0 ± 0.0	**68.0 ± 6.6**
Qwen-OFT	6.0 ± 3.4	0.0 ± 0.0	54.0 ± 7.0	48.0 ± 7.1	0.0 ± 0.0	**66.0 ± 6.7**	14.0 ± 4.9	**14.0 ± 4.9**	54.0 ± 7.0	66.0 ± 6.7	0.0 ± 0.0	56.0 ± 7.0

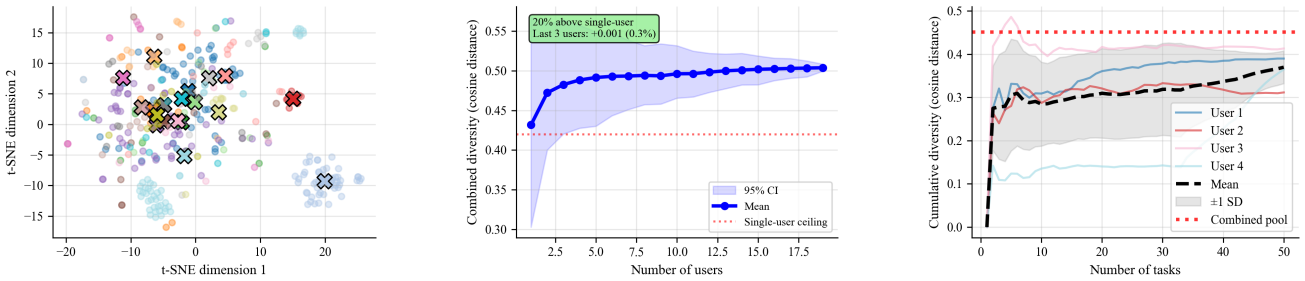
supported tilt toward RoboPlayground over both baselines.

4.2. Evaluating Policies on Training and Generated Generalization Tasks

Tasks. All policies are trained on a shared set of base manipulation tasks covering spatial relations, stacking, alignment, semantic disambiguation, and targeted placement. For each training task, we generate successful demonstration trajectories using CuTAMP [27] and hold the resulting dataset fixed across policies, ensuring that performance differences reflect policy behavior rather than differences in supervision. Evaluation is performed on (i) tasks drawn from the training distribution, and (ii) user generated generalization tasks that require adaptation beyond the training distribution. Generalization tasks are constructed by applying controlled modifications to base tasks using RoboPlayground, including semantic changes to language instructions, visual changes to object attributes and initial configurations (e.g. partially stacked versus scattered on table), and behavioral changes that alter the required action sequences or temporal structure. These transformations follow the task taxonomy described in [6], and are designed to isolate specific dimensions of generalization while preserving task validity. **Models.** We evaluate six policies spanning different action head architectures and finetuning strategies. Four are built on a shared Qwen3-VL-4B-Instruct [4] vision-language backbone and differ in their action decoding mechanism: `Adapter` appends learnable action query tokens to the VLM sequence and decodes actions via an MLP-ResNet regression head with an L1 objective; `GR00T` conditions a flow-matching diffusion transformer (DiT) on the VLM's final hidden states, adopting a dual-system architecture inspired by GR00T N1.5 [24]; `Dual` extends this flow-matching action head with a secondary DINOv2 [25] visual encoder whose patch features are concatenated with the VLM hidden states before conditioning; and `Qwen-OFT` regresses actions from special action-token positions via an MLP head, following the OpenVLA-OFT design [15]. As external baselines, we include `Pi-0.5` [9], a proprietary VLA with flow-matching action generation, evaluated both with full finetuning and with low-rank adaptation (`Pi-0.5 (LoRA)`). All models are trained end-to-end on identical demonstration data. Details of training configurations and generalization tasks are outlined in the appendix.

Training-distribution performance. We first report performance on tasks drawn from the training distribution (top section of Table 2) to provide context for subsequent generalization results. All models achieve moderate to high success on tasks involving simple spatial relations and single-object placement, with `GR00T`, `Dual`, and `Qwen-OFT` consistently outperforming `Pi-0.5` and `Adapter` on most placement and relational tasks. `GR00T` achieves the highest overall in-distribution performance, reaching 96% on `Place Two Blocks on Patch` and leading on stacking-related tasks. However, all models exhibit a consistent difficulty gradient: performance degrades sharply on training tasks requiring greater compositional structure or longer-horizon execution, such as multi-block stacking and color block alignment, where even the strongest model does not exceed 22%. `Adapter` shows notably uneven in-distribution performance, achieving competitive results on some tasks (e.g., 78% on patch placement) while lagging substantially on others (e.g., 26% on left placement). The `Pi-0.5 (LoRA)` variant underperforms all other models across nearly every training task, suggesting that low-rank adaptation alone is insufficient to retain the base model's capabilities in this setting.

Generalization results reveal a clear asymmetry across perturbation types. Across held-out evaluation tasks, all models generalize unevenly across perturbation axes, though the degree of degradation varies by architecture. Performance remains relatively strong under visual perturbations that alter perceptual attributes while preserving execution structure: `GR00T` achieves 90% on `Place Two Blocks on Green Patch` and 86% on `Place Two Blue Blocks on Patch`, and `Dual` similarly transfers well on these tasks (72% and 80%, respectively). Semantic perturbations alone yield mixed but non-zero success for most models, with `GR00T` reaching 78% on `Yellow Block Left Placement` and 62% on `Green on Blue Stack`, and `Dual` achieving 74% and 56% on the



(a) t-SNE of task embeddings by user. (b) Diversity vs. number of users. (c) Cumulative diversity per user.

Figure 4. **Inter-user and intra-user diversity of natural-language manipulation tasks. (a)** t-SNE of sentence embeddings colored by user (crosses = centroids). Tasks cluster by author, indicating systematic conceptual differences. **(b)** Mean pairwise diversity increases monotonically with number of users (shaded: 95% CI). **(c)** Intra-user diversity plateaus quickly; the combined pool (red dashed) achieves substantially higher diversity, highlighting complementary contributions.

same tasks, indicating meaningful robustness to relational re-specification. `Adapter`, however, largely fails under semantic perturbation (e.g., 4% on `Yellow Block Left Placement`, 0% on `Green on Blue Stack`), suggesting that adapter-based finetuning may overfit to surface-level task features. In contrast, tasks involving behavioural perturbations consistently expose severe failure modes across *all* architectures. Tasks requiring multi-stage execution, non-monotonic progress (e.g., unstack-restack), or compositional sequencing yield near-zero success universally—no model exceeds 2% on `Yellow on Red Unstack Restack` or `Stack Two Blocks on Patch`, and `Blue Block Stacking` elicits 0% across the board. These failures occur despite reasonable performance on simpler stacking or placement tasks in isolation, suggesting limited procedural and compositional generalization rather than a lack of basic manipulation competence. `Pi-0.5 (LoRA)` further degrades performance across nearly all perturbation axes, confirming increased sensitivity to deviations from training task structure under constrained adaptation. Together, these results establish behavioural perturbations as the dominant source of generalization failure *independent of model architecture* and motivate evaluation protocols that explicitly probe execution structure.

Implications. Together, these results demonstrate that success on a fixed set of training-distribution tasks can substantially overestimate a policy's robustness. By enabling the generation of creative task variants that probe specific semantic, visual, and behavioral dimensions of generalization, RoboPlayground enables fine-grained diagnosis of policy capabilities and failure modes that are obscured by static task definitions.

4.3. Scalability and Diversity of RoboPlayground

For evaluation, diversity matters not as raw task count, but as coverage of distinct task intents and constraint combinations that probe different policy behaviors. We evaluate how task diversity in RoboPlayground scales with the number of contributors and the number of authored tasks. Each contributor authors up to 50 valid manipulation tasks in the blocks domain using the same interface and asset set. To quantify diversity, we compute average pairwise distances between task representations using semantic sentence embeddings, and analyze both pooled task sets across contributors and cumulative task sets authored by individuals. In the appendix, we report additional analyses using alternative diversity measures, which show consistent trends.

Inter-user diversity scales with contributors. Figure 4(a) shows the distribution of tasks authored by individual users. A t-SNE projection reveals that some users occupy distinct regions of the embedding space, while others exhibit substantial overlap, suggesting systematic differences in how contributors conceptualize and describe manipulation goals within the domain. When tasks are pooled across users (10 tasks per user), the mean pairwise diversity increases monotonically with the number of contributors (Fig. 4(b)). Even after substantial saturation, adding the final three contributors yields a consistent, non-zero increase in diversity, indicating that new contributors continue to introduce semantically novel task formulations.

Intra-user diversity exhibits diminishing returns. In contrast, Figure 4(c) shows that when tasks are added incrementally from a single user, cumulative diversity grows rapidly at first but quickly plateaus. This behavior is consistent across most users, suggesting that individual authors often explore a limited region of the task space, potentially shaped by their preferred abstractions, phrasing, and constraint patterns. Even prolific contributors produce increasingly redundant task variations over time.

Complementarity of multiple contributors. Notably, the combined task pool outperforms any individual contributor in terms of cumulative diversity, as showin in Figure 4(c). This gap highlights the complementary nature of crowd-authored task generation: different users introduce distinct semantic concepts, compositional structures, and constraint combinations that are rarely discovered by a single author alone. Qualitative inspection of task clusters confirms the presence of novel formulations of spatial

Table 3. **Ablation Study.** Cumulative addition of pipeline components, starting with all gates disabled per module. All metrics are percentages ($n = 26$); green ↑/red ↓ show change from prior row.

Module	Configuration	Task Succ.	Compile	Smoke Test	Human-Ver.	LLM Align.
Task Proposal	None (all disabled)	100.0	100.0	100.0	88.5	74.0
	+ asset inference	100.0	100.0	100.0	92.3 (↑3.8)	71.5 (↓2.5)
	+ feasibility checking (all gates on)	**100.0**	**100.0**	**100.0**	**100.0** (↑7.7)	**73.6** (↑2.1)
Code Generation	None (all disabled)	100.0	100.0	100.0	96.2	70.8
	+ API review	96.2 (↓3.8)	100.0	100.0	100.0 (↑3.8)	70.5 (↓0.3)
	+ common errors review	100.0 (↑3.8)	100.0	100.0	96.2 (↓3.8)	72.4 (↑1.9)
	+ in-context examples (all gates on)	**100.0**	**100.0**	**100.0**	**100.0** (↑3.8)	**73.6** (↑1.2)
Validation	None (all disabled)	12.0	96.0	12.0	96.2	71.1
	+ text validation	96.2 (↑84.2)	100.0 (↑4)	100.0 (↑88)	96.2	73.8 (↑2.7)
	+ compilation	100.0 (↑3.8)	100.0	100.0	100.0 (↑3.8)	71.5 (↓2.3)
	+ instantiation runtime	100.0	100.0	100.0	96.2 (↓3.8)	72.3 (↑0.8)
	+ success checking	96.2 (↓3.8)	100.0	96.2 (↓3.8)	96.2	73.0 (↑0.7)
	+ bounds checking	100.0 (↑3.8)	100.0	100.0 (↑3.8)	92.3 (↓3.9)	73.1 (↑0.1)
	+ specialist agents (all gates on)	**100.0**	**100.0**	**100.0**	**100.0** (↑7.7)	**73.6** (↑0.5)
Context Steering	None (all disabled)	95.7	100.0	100.0	92.3	73.7
	+ intent interpretation	96.2 (↑0.5)	100.0	96.2 (↓3.8)	100.0 (↑7.7)	73.8 (↑0.1)
	+ routing classification	96.2	100.0	96.2	76.9 (↓23.1)	72.5 (↓1.3)
	+ version history tracking	96.2	100.0	96.2	100.0 (↑23.1)	73.6 (↑1.1)
	+ reference selection (all gates on)	**100.0** (↑3.8)	**100.0**	**100.0** (↑3.8)	**100.0**	**73.6**

relations, multi-object constraints, and success conditions that are absent from single-author collections.

Overall, these results show that RoboPlayground scales not merely by increasing task count, but by expanding coverage of the underlying task space through contributor diversity. By expanding coverage across intent and constraint structure, RoboPlayground enables evaluation to reveal brittleness to even seemingly minor semantic variations.

5. Ablative Studies

We conduct a cumulative ablation study to quantify the functional contribution of each component in the task generation pipeline. For each module, we begin with all components disabled and progressively enable individual gates. This design disentangles changes in semantic task specification from improvements in robustness and correctness under session-level evaluation (Table 3).

Metrics and Evaluation Setting. We report complementary metrics capturing distinct failure modes. *Compile* and *Smoke Test* measure code correctness and execution stability; *Task Success* measures end-to-end satisfaction of the success predicate; *Human Verification* evaluates perceived task validity; and *LLM Alignment* measures consistency between the natural language instruction and the implemented success condition. Ablations are evaluated on ten benchmark testcases, each consisting of multiple related tasks evaluated as a single session; some testcases involve multi-stage task refinement via context-aware steering.

Task Proposal. Task proposal components primarily affect semantic grounding rather than executability. Enabling asset inference improves Human Verification (88.5 to 92.3) but slightly reduces LLM Alignment (74.0 to 71.5), while leaving execution metrics unchanged. Adding feasibility checking improves both Human Verification (92.3 to 96.2) and LLM Alignment (71.5 to 73.6) without affecting executability.

Code Generation. Code generation components primarily improve robustness to systematic implementation errors. Across ablations, compilation and smoke test success remain near-perfect. API review, error checks, and in-context examples incrementally improve LLM Alignment (70.8 to 73.6) while preserving end-to-end executability.

Validation. Validation is the dominant determinant of task correctness. With validation disabled, Task Success drops to 12.0 despite high compilation rates. Text-level validation alone recovers Task Success to 96.2, while the full validation stack achieves perfect Task Success, Compile, Smoke Test, and Human Verification.

Context Steering. Context steering influences semantic coherence across multi-step task sessions. Intent interpretation and version history tracking improve Task Success, Human Verification, and LLM Alignment, while routing without history degrades semantic consistency. With full context steering enabled, execution metrics remain perfect.

6. Discussions

This work explores how robotic manipulation evaluation changes when task specification is opened to a broader set of contributors. By treating language as an executable interface, RoboPlayground allows users to express task intent, constraints, and success criteria directly, rather than relying on fixed, expert-authored benchmarks. In doing so, it reframes evaluation as a process shaped not only by models and metrics, but by the people defining what is being tested.

References

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

[2] Anysphere. Cursor: Ai-powered code editor. https://cursor.com/, 2023. AI-assisted integrated development environment. Available at https://cursor.com/. 5

[3] Pranav Atreya, Karl Pertsch, Tony Lee, Moo Jin Kim, Arhan Jain, Artur Kuramshin, Clemens Eppner, Cyrus Neary, Edward Hu, Fabio Ramos, et al. Roboarena: Distributed real-world evaluation of generalist robot policies, 2025. 2

[4] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report, 2025. 6

[5] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996. 5

[6] Jensen Gao, Suneel Belkhale, Sudeep Dasari, Ashwin Balakrishna, Dhruv Shah, and Dorsa Sadigh. A taxonomy for evaluating generalist robot manipulation policies. *IEEE Robotics and Automation Letters*, 2026. 6

[7] Ran Gong, Xiaohan Zhang, Jinghuan Shang, Maria Vittoria Minniti, Jigarkumar Patel, Valerio Pepe, Riedana Yan, Ahmet Gundogdu, Ivan Kapelyukh, Ali Abbas, Xiaoqiang Yan, Harsh Patel, Laura Herlant, and Karl Schmeckpeper. Anytask: an automated task and data generation framework for advancing sim-to-real policy learning, 2026. 2

[8] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, pages 139–183. Elsevier, 1988. 5

[9] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. 6

[10] Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, Wei-Chiu Ma, Dhruv Shah, Abhishek Gupta, and Karl Pertsch. Polaris: Scalable real-to-sim evaluations for generalist robot policies, 2025. 2

[11] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment, 2019. 2

[12] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, 2016. 1

[13] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023. 2

[14] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in nlp, 2021. 1

[15] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. 6

[16] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation, 2024. 2

[17] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation, 2024. 2

[18] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2023. 3

[19] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models, 2024. 2

[20] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. 2

[21] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2024. 2

[22] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations, 2021. 2

[23] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots, 2024. 2

[24] NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, et al. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. 6

[25] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision, 2024. 6

[26] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation, 2024. 2

[27] William Shen, Caelan Garrett, Nishanth Kumar, Ankit Goyal,

Tucker Hermans, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Fabio Ramos. Differentiable gpu-parallelized task and motion planning. *arXiv preprint arXiv:2411.11833*, 2024. 6

[28] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023. 2, 5

[29] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation, 2024. 2

[30] Yi Ru Wang, Carter Ung, Grant Tannert, Jiafei Duan, Josephine Li, Amy Le, Rishabh Oswal, Markus Grotz, Wilbert Pumacay, Yuquan Deng, Ranjay Krishna, Dieter Fox, and Siddhartha Srinivasa. Roboeval: Where robotic manipulation meets structured and scalable evaluation, 2025. 2