arXiv:2507.12644v1 [cs.RO] 16 Jul 2025

VLMGINEER: Vision Language Models as Robotic Toolsmiths

George Jiayuan Gao*, Tianyu Li*, Junyao Shi, Yihan Li†, Zizhe Zhang†, Nadia Figueroa, Dinesh Jayaraman
vlmgineer.github.io/release
Email correspondence: {gegao, tianyu}@seas.upenn.edu. * and † denote equal contribution.

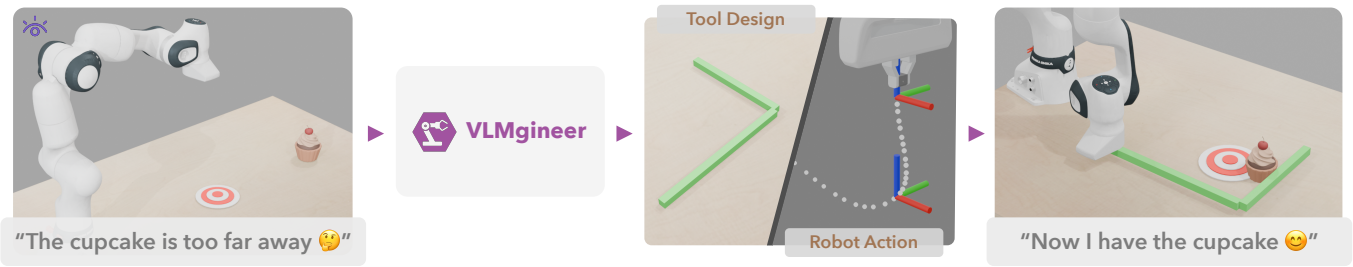


Fig. 1: Given a manipulation task that lies outside the robot's capabilities, **VLMGINEER** first prompts a vision language model to generate a tool and action. We then employ evolutionary search in simulation to refine the tool's geometry and synthesize the corresponding robot motion plan. Finally, the robot, equipped with the automatically designed tool, successfully completes the task.

***Abstract*—Tool design and use reflect the ability to understand and manipulate the physical world through creativity, planning, and foresight. As such, these capabilities are often regarded as measurable indicators of intelligence across biological species. While much of today's research on robotic intelligence focuses on generating better controllers, inventing smarter tools offers a complementary form of physical intelligence: shifting the onus of problem-solving onto the tool's design. Given the vast and impressive common-sense, reasoning, and creative capabilities of today's foundation models, we investigate whether these models can provide useful priors to automatically design and effectively wield such tools? We present VLMGINEER, a framework that harnesses the code generation abilities of vision language models (VLMs) together with evolutionary search to iteratively co-design physical tools and the action plans that operate them to perform a task. We evaluate VLMGINEER on a diverse new benchmark of everyday manipulation scenarios that demand creative tool design and use. Across this suite, VLMGINEER consistently discovers tools and policies that solve tasks more effectively and innovatively, transforming challenging robotics problems into straightforward executions. It also outperforms VLM-generated designs from human specifications and existing human-crafted tools for everyday tasks. To facilitate future research on automated tool invention, we will release our benchmark and code. Project Website: vlmgineer.github.io.**

I. INTRODUCTION

Humans exhibit a remarkable ability to design and utilize tools, fundamentally extending their capabilities to accomplish tasks otherwise beyond their reach through creativity, planning, and foresight. This capacity for tool creation and usage represents one of our most distinctive cognitive adaptations, and therefore is widely regarded as a marker of cognitive complexity.

Much of today's robotics research concentrates on enabling complex robot motions to utilize standard pre-designed robot end-effectors or tools [1–5]. In this work, we pursue an alternative form of physical intelligence: inventing tools that simplify downstream control, thereby shifting the primary problem-solving burden from devising control strategies to designing the tool's geometry. This requires solving a complex co-design problem over tool designs and the motions that wield those tools. The search space is dauntingly large, and it is clear that the design procedure would benefit from suitable priors to guide this search.

One potential source of such priors might be large vision-language models (VLMs) pre-trained on web-scale data. State-of-the-art VLMs have been shown to possess impressive common-sense, reasoning, and creative abilities, alongside extraordinary capabilities in code generation, visual comprehension, and in-context learning. When combined with evolutionary search methods, VLMs have successfully crafted human-level reward functions for reinforcement learning [6, 7], 3D graphics [8], articulations of in-the-wild objects [9], intricate 3D sculptural designs [10], and developing advanced algorithms to solve mathematics and science problems [11–13].

So, *can today's VLMs also guide the design of innovative and action-efficient physical tools for robots*? We introduce VLMGINEER, a fully autonomous framework that leverages VLMs to jointly evolve both tool design and manipulation strategies for robots. Our method demonstrates unprecedented efficacy in developing specialized tools for diverse manipulation tasks, through an evolutionary search process guided by VLM-generated tool geometries and action plans. Compared to prior

more limited investigations of tool design, that largely consider parameter optimization for a manually designed parametric template, VLMGINEER works off-the-shelf for new tasks without task-specific templates, prompts, or examples. Furthermore, compared to prior works that use VLMs, VLMGINEER does not require in-context few-shot examples. To facilitate future research and benchmarking, we also introduce ROBOTOOLBENCH, a comprehensive simulation suite comprising 12 diverse robotic tool-use manipulation tasks specifically designed to evaluate tool design and policy optimization methods.

Our fully autonomous approach demonstrates superior task performance over designs generated with human specifications and human-crafted everyday tools. When evaluated on ROBOTOOLBENCH, VLMGINEER achieves an average normalized improvement of 64.7% over VLM-generated designs from human language specifications and outperforms existing human-crafted tools by an average normalized improvement of 24.3%. Our results serve to validate both the physical design intelligence enshrined in VLMs pre-trained on web-scale data, and also present the promise of more adaptable and capable robotics systems that can ingeniously create and use tools.

II. RELATED WORK

Task-Specific Computational Agent and Tool Design. Previous research has extensively investigated methods for optimizing robot morphology, end-effectors, and tool designs for robot manipulation through various computational approaches, ranging from model-based optimization [14], reinforcement learning (RL) [15–17], data-driven generative models [18–21], and differentiable simulation [22]. Others have explored robot design for locomotion using evolutionary algorithms [23–28], stochastic optimization [29], and graph search [30]. However, these existing approaches typically require manual task-specific pre-definition of a handful of optimization parameters, rely on fixed trajectories or pre-defined control policies, and tend to suffer from low sample efficiency. In contrast, we introduce a VLM-driven approach that simultaneously optimizes both tool design and manipulation strategy, enabling generalization across diverse manipulation tasks without requiring manual parameter specifications.

Robot Learning for Tool-Based Tasks. To learn effective tool usage, some have employed learned or simulated dynamics models for tool manipulation optimization [31–35]. Another prevalent approach involves learning tool and object affordances — understanding the functions of objects and tool-object interactions [1, 36–40]. Recently, large language models have been leveraged for creative tool use [41]. While these methods all assume that suitable tools already exist in the environment, we instead address the more challenging scenario where a general-purpose robot must concurrently invent the right tool design and the appropriate way to use it for a task.

Joint Optimization of Morphology and Control. A common strategy for jointly addressing tool design and manipulation problems is to formulate them as nonlinear programs within a predefined parameter space including both design and manipulation variables, a method that has demonstrated particular effectiveness in sequential manipulation tasks involving extended horizons [42, 43]. However, given our objective to deploy VLMGINEER in any arbitrary environment without manual specification of design parameters, we rely on the underexplored physical creativity of VLMs. Approaches using RL [44–47], gradient-based optimization [48], Bayesian optimization, evolutionary algorithms [49–51], or a combination of them [52–56] have been proposed for joint morphology and control learning for robot locomotion tasks, in particular with soft or modular robots. Studies on joint robot or tool and policy design through RL [57, 58], differentiable simulation [59], and model-based optimization [60] have also demonstrated effectiveness in tool manipulation. However, since these methods still all require manual specifications of the design space, they require significant human efforts to scale beyond a few tasks.

Recent work has explored LLM-aided evolutionary search for robot design in conjunction with RL-based policy optimization in locomotion [61–63], demonstrating the potential of using LLMs to unlock more performant robot design. Unlike prior work, our work targets open-world VLM-guided design of both tools and actions for manipulation without human-in-the-loop parameter specification. **VLMGINEER leverages the surprising physical intuition and creativity of VLMs to automatically create design solutions using evolutionary search.** It can easily be scaled to a wide range of tasks, and it is much more efficient in terms of samples, time, and compute than prior RL-based methods.

III. BACKGROUND

The goal of VLMGINEER is to jointly optimize both the physical design of tools and the manipulation strategies needed to accomplish a wide range of robotic tasks, without requiring human-specified design parameters or fixed control policies. At its core, VLMGINEER is an evolutionary search algorithm, enhanced by the creative and commonsense reasoning capabilities of vision–language models, to efficiently discover novel tool–action pairs for challenging manipulation problems.

Evolutionary Methods. Evolutionary algorithms [64, 65] have a long-standing history in solving optimization problems, inspired by principles of biological evolution and natural selection. They are particularly effective in black-box optimization with vast optimization spaces, such as open-ended design. At their core, these methods maintain a **population** of candidate solutions, which iteratively evolve through carefully designed mutation and crossover operators. Each iteration evaluates individuals against a **fitness function**, selecting those with higher fitness while discarding or replacing less successful candidates. To balance exploitation and exploration, **crossover** combines promising solutions into offspring, and **mutation** introduces novel variations. Evolutionary algorithms have proven effective across diverse domains such as program synthesis, symbolic regression, algorithm discovery, and even robot design. Nevertheless, their reliance on handcrafted mutation and crossover operators remains a significant limitation—such

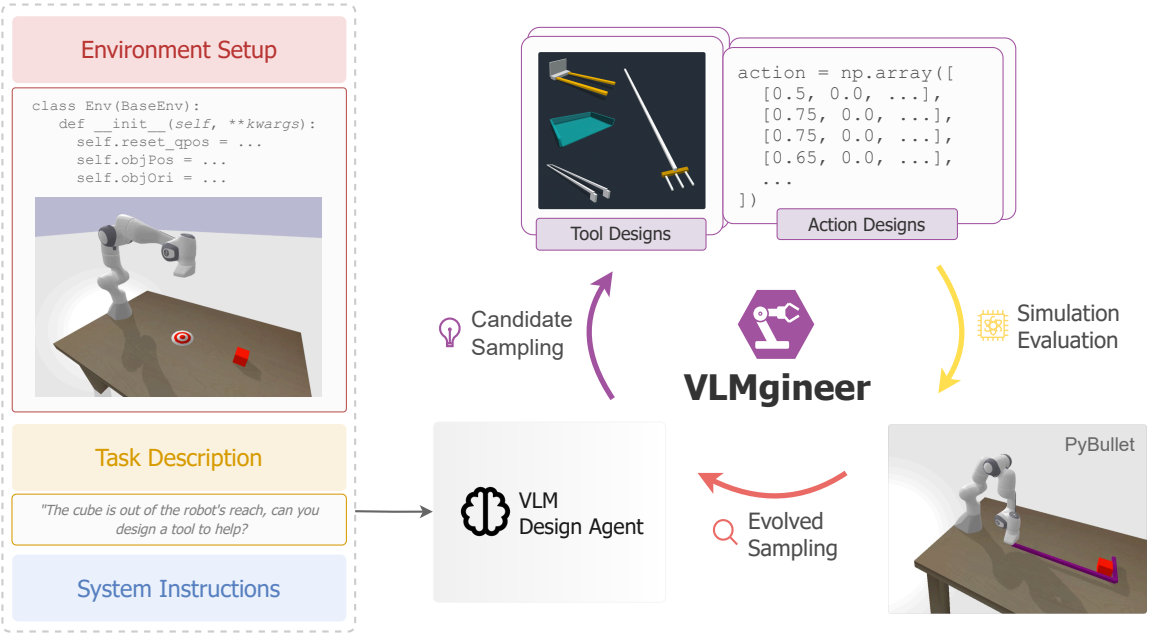


Fig. 2: VLMGINEER takes unmodified environment source code, environment image, environmental description, and task description as context to zero-shot generate tool and action designs from a VLM. It then iteratively refines its tool and action designs through a loop of candidate sampling, simulation-based evaluation, and evolution improvement.

Algorithm 1 VLMGINEER: Evolutionary Tool and Action Co-Design with VLMs

Require: Environment code $\mathcal{E}$, image render I, task description d_{task}, fitness function $\mathcal{F}$, initial prompt PROMPT, Vision-Language Model VLM
Hyperparameters: Number of evolution cycles n, population size K, top-k selection threshold
for n iterations do **do**
  `// Sample K designs`
  $D_1, D_2, ..., D_K \sim \text{VLM}(\mathcal{E}, I, d_{\text{task}}, \text{PROMPT})$
  `// Evaluate design candidates`
  $s_1 = \mathcal{F}(D_1), \cdots, s_K = \mathcal{F}(D_K)$
  `// Selection`
  Select top-k designs $\{D_{j_1}, ..., D_{j_k}\}$ with highest s_j
  `// Evolution`
  PROMPT := PROMPT : EVOLVE_PROMPT$(\{D_{j_1}, ..., D_{j_k}\})$
end for
return Final design $D^* = \arg\max_D \mathcal{F}(D)$ across all iterations

operators are challenging to design and often inadequately capture essential domain-specific insights.

Large Model-Guided Evolution. To improve the scalability, performance, and automation of evolutionary algorithms, recent work has integrated large models into the evolutionary process, automating initial population generation, mutation and crossover operations. Leveraging the extensive world knowledge and inductive biases inherent in large models allows for more efficient evolution of candidate solutions and also eliminates the necessity of manually defining allowed mutation operations. Moreover, some approaches exploit the rich semantic understanding of large models to provide nuanced, semantic feedback beyond simple numerical fitness scores. Specific implementations of these principles in evolutionary algorithms vary according to the domain. For instance, Eureka [7] employs large language models (LLMs) to guide evolutionary reward design in reinforcement learning.

Eureka generates a *population* of candidate reward functions directly from raw environment code, evaluates RL agents trained with these rewards using a task-specific *fitness* function, and selects the best-performing candidates. Although it omits explicit crossover, Eureka employs LLM-guided in-context reward *mutation* by proposing an improved reward function from an existing one based on textual feedback.. Drawing inspiration from these successes, we investigate whether vision–language models (VLMs) can similarly offer valuable inductive biases to guide the evolutionary design of robotic tools and manipulation actions.

IV. METHOD

VLMGINEER builds upon existing Large Model-guided evolutionary methods to achieve effective tool-action co-design. Specifically, VLMGINEER comprises three algorithmic components: **(1) Population Generation**: We query the VLM to generate a diverse population of candidate designs D_i (tool-action tuples) using the raw environment code $\mathcal{E}$, an image render I, task description d_{task}, and an initial prompt PROMPT as context. **(2) Fitness Evaluation**: We assess each candidate design D_i using task-specific fitness functions $\mathcal{F}$, retaining only the top-k designs based on their computed rewards. **(3) Iterative Evolution**: We iteratively prompt the VLM to produce novel offspring designs through guided mutations and crossovers, progressively enhancing tool-action quality.

Joint Tool and Action Candidate Sampling. While previous approaches of large model-guided evolutionary robot

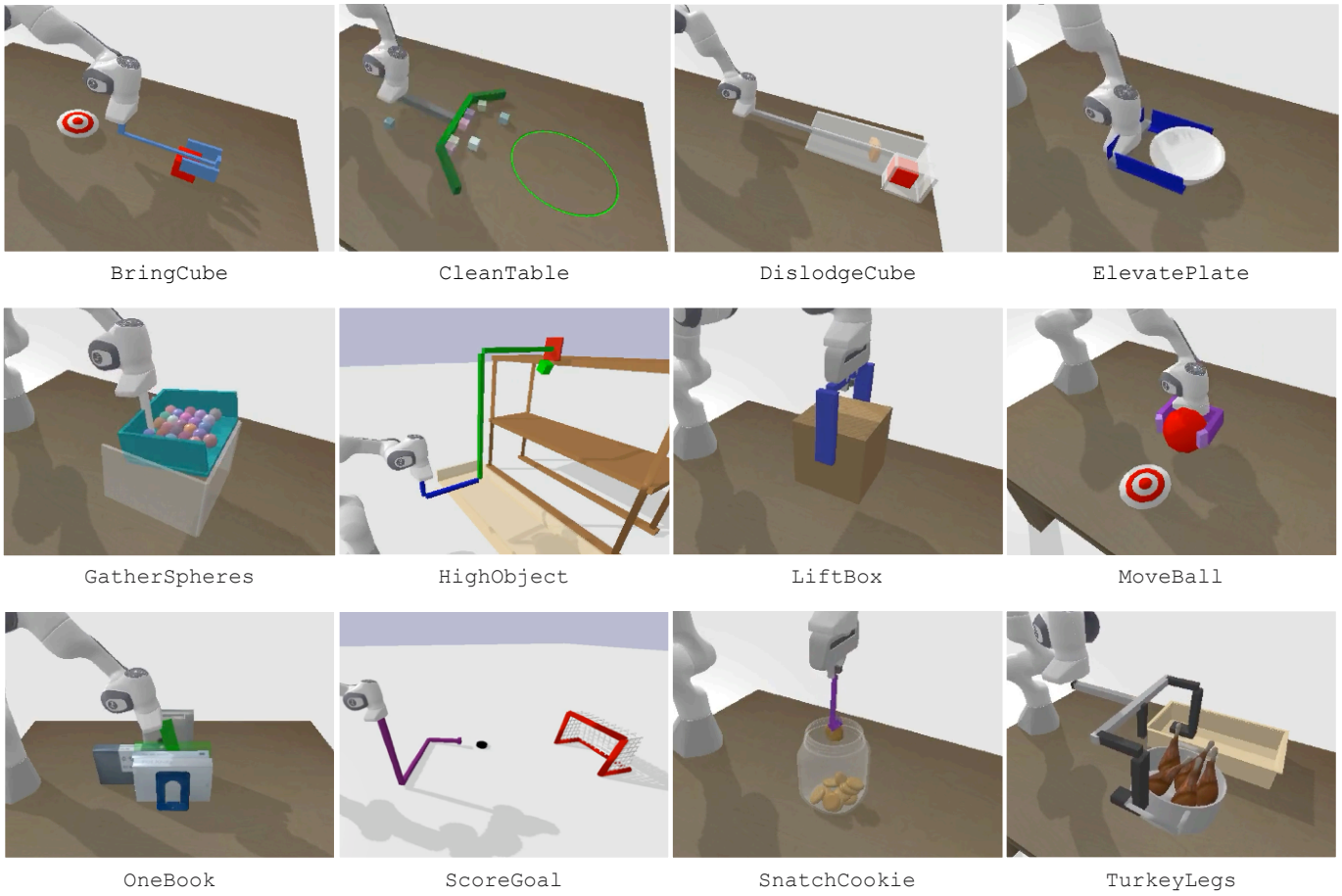


Fig. 3: VLMGINEER produces innovative tool designs and their corresponding actions across 12 diverse tasks in ROBOTOOLBENCH that are challenging to perform using a general-purpose robot arm and gripper.

design [61, 63] typically optimize robot morphology alone, relegating action or control optimization to a subsequent evaluation stage. our approach prompts the VLM to simultaneously generate paired tool designs and corresponding action strategies in a single inference step. Joint tool-action sampling allows for a tighter coupling between tools and their associated actions. Rather than sequentially optimizing the tool geometry first and then actions afterward, simultaneous optimization leverages the VLM's inductive biases to smoothly navigate the joint tool–action design space towards the Pareto frontier. Concretely, within each evolution cycle, VLMGINEER prompts the VLM to propose n distinct tool designs along with m candidate action plans per tool, resulting in $n \times m$ total tool-action pairs. This corresponds to a kind of crude VLM-guided policy optimization, which merely selects the best among the m generated action plans. Compared to policy optimization via RL [7, 61, 63], our action sampling approach, albeit simple, significantly accelerates iteration cycles and reduces computational overhead by exploiting the insight that appropriately designed tools inherently simplify and enhance action plans.

Specification of How to Do Crossover And Mutation. A critical part of how VLMGINEER enables effective tool design evolution is the utilization of *inductive in-context* ***crossover and mutation***. We define inductive in-context crossover and mutation as the process of prompting VLMs to introduce random, free-form tool mutations and crossovers, conditioned on previous elite tool candidates, and guided by the model's learned inductive biases for producing better task-solving tools. We use the prompt below to perform inductive in-context crossover and mutation: *Your design decision is part of a genetic algorithm for tool creation, where each new design is produced either by mutation—changing exactly one aspect (e.g., adjusting a component's dimension or adding/removing a component)—or by crossover, combining elements from two existing designs. All resulting mutations and crossovers should plausibly enhance task success while preserving design diversity.*

Tool Representation Format. Selecting an appropriate representation for tools—balancing abstraction, design flexibility, and manufacturability—is critical for effective optimization. Prior works have represented objects and tools as meshes [66], CAD [67], or blocks [10]. These representations, however, either introduce excessive complexity and optimization challenges or lack sufficient expressiveness. Inspired by prior

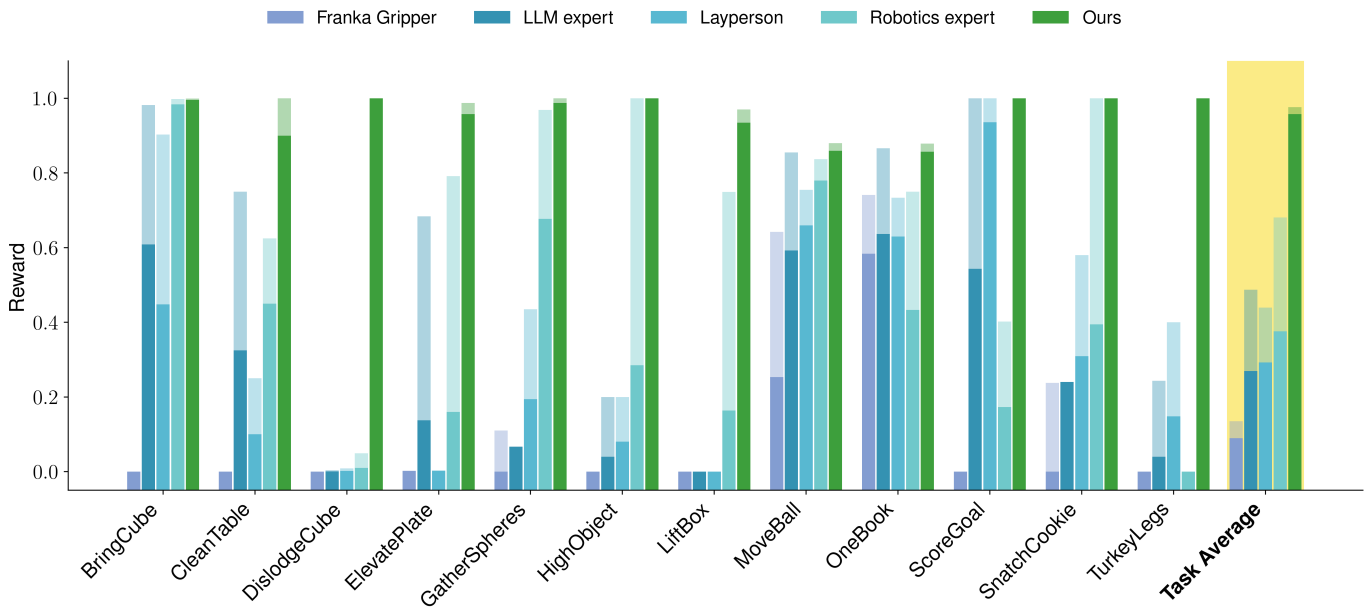


Fig. 4: This figure compares the reward of Franka Gripper experiments, 3 Human Prompt experiments, and experiments on our proposed method across 12 tasks. For every method, the bars with the original dark color in the legend indicate the *average* reward of the five runs, while the bars with a paler color visible above them indicate the *best* reward over those runs.

work [9], we represent tools in Unified Robot Description Format (URDF). The structured, modular nature of URDF, analogous to code blocks, aligns seamlessly with vision–language models' (VLMs) strengths in code understanding and generation. Concretely, we prompt the VLM to generate URDF-defined tool designs as modular blocks that can be directly integrated into a designated end-effector link of the robot model.

Action Representation Format. Building on recent work that leverages VLMs for action generation [68, 69], we prompt the model to explicitly output action sequences in the form of an $N \times 7$ array, where N denotes the number of waypoints. Each row encodes a 6-DoF pose for the robot end-effector, along with a gripper open/close command.

V. Robot Tool Design Benchmark

We propose a comprehensive simulation benchmark **RoboToolBench** designed explicitly for evaluating robotic tool and policy design. RoboToolBench comprises 12 object manipulation tasks designed to be challenging for the base robot arm to complete without any tools. These task environments are visualized in Fig. 3. For several tasks (`BringCube`, `CleanTable`, `GatherSpheres`, `ScoreGoal`), we took inspiration from the subset of RLBench [70] tasks that involve tool use — note, however, that we expect that automated tool design will *replace and improve* the original tools from RLBench. Several other tasks (`HighObject`, `ElevatePlate`) are inspired by prior works in computational co-design [58] that study task-specific design parameter optimization as discussed in Sec II. Still more task environments are inspired by everyday home scenarios (`LiftBox`, `MoveBall`, `OneBook`, `SnatchCookie`, `TurkeyLegs`). Finally, `DislodgeCube` is inspired by a tool design behavior previously observed in the Caledonian crow [71], which used tools to retrieve objects in confined spaces. We adopt the Franka Panda robot arm as the standard morphology to attach tools to, and implement our environments using PyBullet [72].

VI. Evaluation

We begin this section with an introduction to our experimental goals and setups, and then analyze the results of our comparison and ablation studies in detail. Our experiments are designed to address the following questions: **Q1:** Can VLMgineer effectively discover innovative tools and ways to use them? **Q2:** How does VLMgineer compare to a human specifying tool designs to a VLM in natural language? **Q3:** How do VLMgineer outputs improve over evolution iterations?

a) Baselines.: To showcase VLMgineer's ability to generate creative and effective tools and usage actions, we compare our method with the following baselines: **(1) Franka Gripper**: We evaluate the performance of the vanilla Franka Panda two-finger gripper without additional tools on RoboToolBench to highlight the inherent limitations of the robot's default morphology; these tasks are after all explicitly designed to be very hard or impossible to perform without the right tools. We derive the no-tool action policy by prompting the VLM to follow an action-sampling procedure analogous to our proposed method, minus the use of any tools. **(2) Human Prompts**: For these baselines, we ask humans to specify a tool design to the VLM in natural language, following which it attempts to generate that tool and several action plans, as in our method. There is no evolutionary search. We evaluate on humans

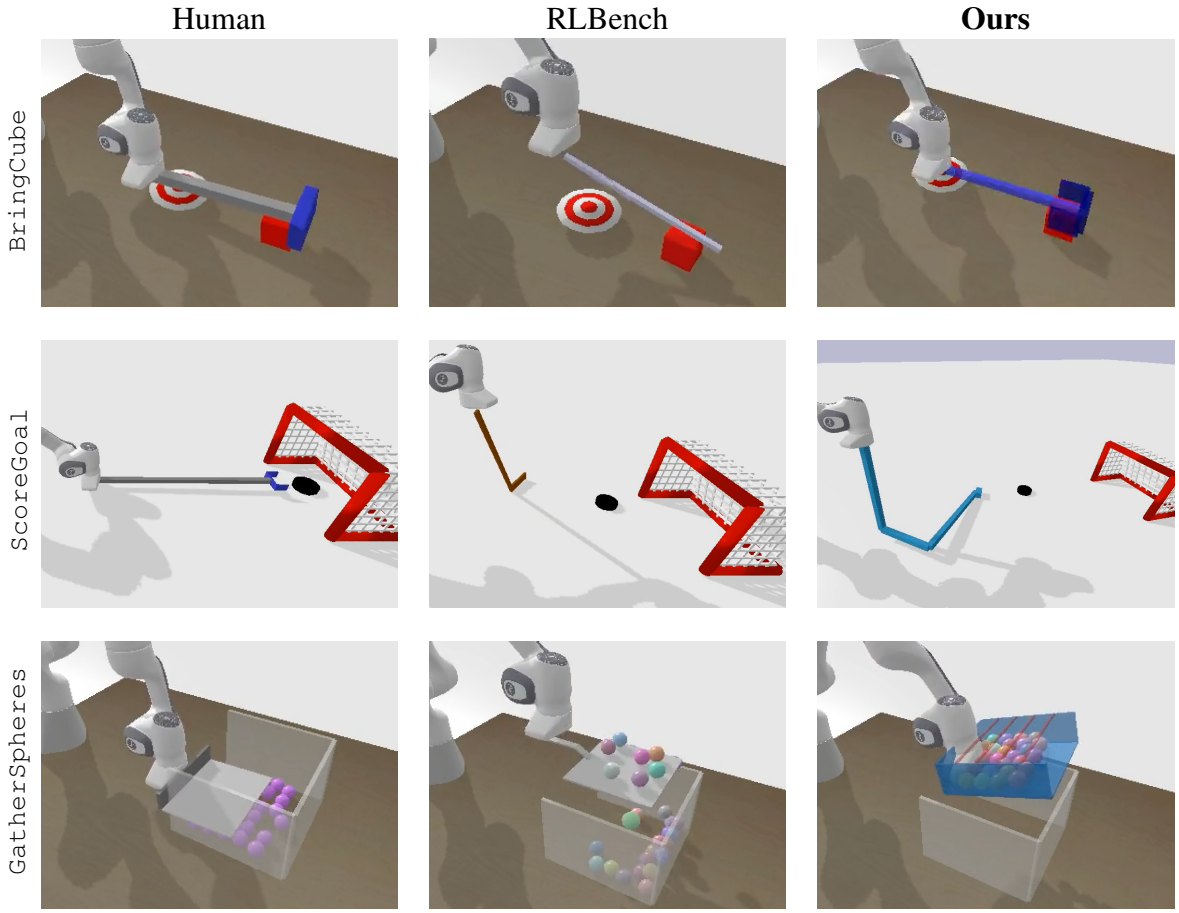


Fig. 5: Qualitative comparison of human-designed, RLBench, and VLMGINEER tools on three tasks: `BringCube` (top row), `ScoreGoal` (middle row), and `GatherSpheres` (bottom row).

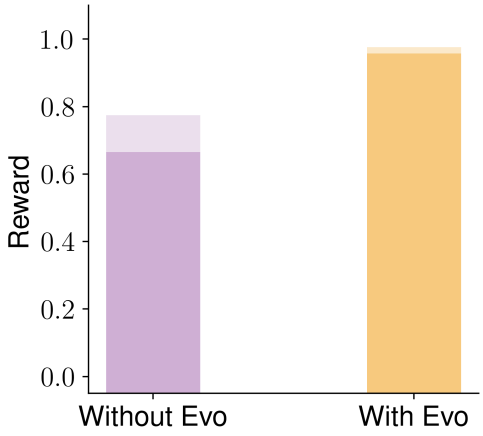
with varying expertise: "Robotics expert" (a graduate student researching robot learning), "LLM expert" (a graduate student researching LLMs), and "Layperson" (an undergraduate student with no relevant research experience). The procedure on the case study is in Appendix A.1. **(3) RLBench Tools**: We evaluate four original tools from the tasks we adapted from RLBench, which are often natural everyday tools for the tasks considered.

b) Evaluation Metrics.: To assess the quality of a tool-action design after each execution, we define these evaluation metrics: **(1) Task Reward**, which is a set of pre-defined task reward function $R : S \to r \in [0, 1]$ that are unique to each task, where S is its environmental state and r is a normalized reward. These rewards are designed to evaluate the progress made in the task by a certain tool-action pair. **(2) Distance Traversed**, defined as the total distance traversed (in meters) by the robot end-effector for completing a tool-action pair execution to evaluate the effort efficiency of the design. This is motivated by how better tools tend to reduce the effort needed to complete a task.

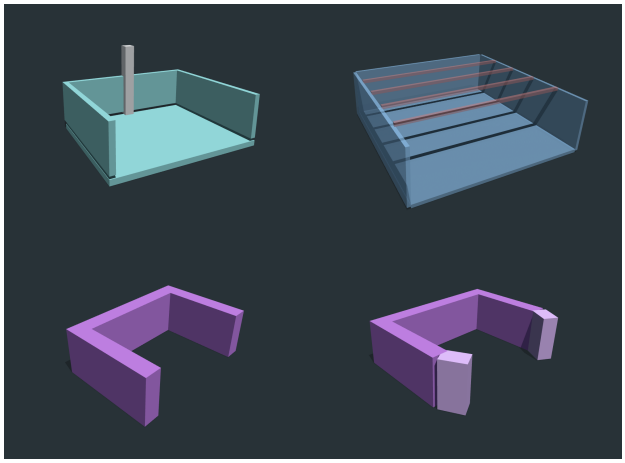
We run each baseline approach and our method five times on each task, then report the best and average rewards across those five runs. To more clearly differentiate experiments that resulted in similar rewards, we use the distance traversed as a secondary tie-breaker metric that rewards efficiency.

A. *Comparing* VLMGINEER *Results to other Tool Designs*

The results are summarized in Fig. 4. VLMGINEER works consistently well across tasks, in terms of both average and best rewards. We dive into interesting individual method comparisons now. As expected, the default Franka Panda two-finger gripper fails on the majority of these tasks. What is perhaps more noteworthy is that **VLMGINEER outperforms human-prompting!** This is true even for expert humans across all tasks and on both metrics (better peak performance and also more reliable). While human prompts occasionally produced strong solutions, their results were less consistent and efficient. In tasks like `CleanTable` and `ScoreGoal`, both approaches reached similar peak rewards, but our method did so with significantly shorter paths. For further analysis, Fig. 5 shows example designs from human-prompting and VLMGINEER. Human-designed tools (left column) generally offer suitable forms for task completion; however, VLMGINEER (right column) creates more specialized features that enhance performance. For instance, in task `ScoreGoal`, our method produces long and bent shapes facilitating simpler, more efficient motions,



(a) Mean top reward across all evaluated tasks, comparing the Without/With Evo condition.



Before Evo. After Evo.

(b) Qualitative comparison before/after evolution on `GatherSpheres` (top row) and `MoveBall` (bottom row).

Fig. 6: We present the quantitative (a) and qualitative (b) effectiveness of evolution in tool design.

which the robot just need to move very little along one axis to hit the puck. On the other hand, the straight tool designed from human prompts would require more careful control of the puck. In `GatherSphere`, our design includes a scoop with side protection and an overhead stripe structure, effectively preventing spheres from bouncing away.

VLMGINEER tools also outperform the RLBench original tools. On the four RLBench-based tasks in ROBOTOOLBENCH, we evaluated the standard RLBench Tools (Fig. 5 middle column). These are often designed to be simple everyday tools that humans might use for those tasks. As shown in Fig. 7, across every task, VLMGINEER not only attains the highest possible reward across the repeated five runs but does so more reliably (on the average reward) than RLBench Tools. While the rewards in task `BringCube` and `CleanTable` are similar to ours, the average rewards over the five repeated runs are generally lower than VLMGINEER. Given that the best rewards over five repeated runs are similar in `BringCube` and `CleanTable`, we use distance traversed as the tie breaker. From Fig. 7, corresponding to the best reward, our method has a lower distance in `BringCube` and almost the same distance in `CleanTable`.

Quantitatively inspecting the tools further highlights the advantages of our method over the original tools in RLBench. The RLBench tools (middle column), originally designed for similar but distinct tasks, often underperform due to less optimized features. For example, in `BringCube`, the RLBench's simple stick, from the original *reach and drag*, provides insufficient lateral control, resulting in inconsistent cube manipulation. Our method's cage-like structure reliably locks and moves the cube closer, achieving significantly higher rewards. Similarly, in `ScoreGoal`, RLBench's hockey stick, from their *hockey* task, demands precise, extensive movements, whereas our geometrically optimized tool scores easily with minimal end-effector movement. In `GatherSpheres`, RLBench's spatula, from their *scooping with a spatula* task, lacks effective side control, causing frequent sphere roll-offs, while our design with protective edges achieves more successful scoops and higher task rewards. It is important to note, however, that RLBench tools are not necessarily designed to maximize task ease or performance. Rather, they are chosen to reflect the common tools used by humans for these tasks as a means of benchmarking manipulation policies.

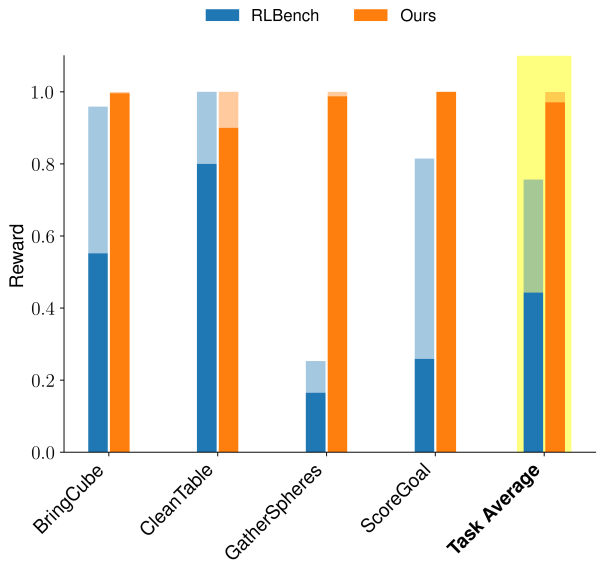


Fig. 7: This figure shows the reward of experiments with the original tools in RLBench compared with the experiments on our proposed method across 4 tasks, in which the bars with darker color indicate the average best reward across the five runs, while the bars with paler colors indicate the best reward across the five runs.

B. Effectiveness of Evolution in Tool Design

To quantify the necessity of the evolutionary optimization within VLMGINEER, we conducted an ablation study comparing our full evolutionary framework against a sampling-only baseline (**VLMGINEER w.o. Evolution**), which performs initial tool and action generation without iterative improvement. Fig. 6a illustrates the best and average rewards achieved by both configurations across selected tasks.

The results clearly demonstrate the efficacy of evolutionary refinement. VLMGINEER consistently achieves higher task rewards, showcasing its capability to explore and identify superior regions in the joint tool-action design space that remain inaccessible through initial sampling alone. This improvement underscores the value of iterative evolutionary optimization.

To gain qualitative insights into how evolution incrementally enhances designs, we visualize examples from the evolutionary process in Fig. 6b. In the `GatherSpheres` task, the initial scooping tool lacked coverage at the top, allowing spheres to bounce out frequently. Evolutionary iterations addressed this by adding guardrails, significantly improving containment and task success rates. Similarly, in the `MoveBall` task, the original tool's open-ended design made ball handling challenging. Evolution optimized the geometry by introducing a hugging rim, greatly enhancing control and maneuverability.

These qualitative examples, together with the quantitative evidence, confirm that the evolutionary component of VLMGINEER not only refines initial designs but is critical for achieving robust and effective solutions that maximize performance.

VII. Conclusion

We propose a new framework for co-optimizing tool design and tool use actions by leveraging the creativity from VLM. By evaluating on 12 different simulation tasks, we demonstrate the capability to design and use tools to solve robotic manipulation tasks. Our results show that we outperform baselines that (1) don't design or use tools and (2) take the specifications directly from humans. We also perform an ablation study to show how our evolutionary module could further boost the performance.

Limitations. While VLMGINEER demonstrates significant advancements in robotic tool and action co-design, several limitations remain that future research should address: (1) Our current framework relies exclusively on simulated environments, potentially impacting the real-world effectiveness and transferability of generated designs. (2) Robot actions are represented as discrete end-effector poses, limiting the handling of complex dynamic tasks requiring precise temporal coordination. (3) Tool representations in URDF format are constrained to simple geometries and limited material properties, and while preliminary results suggest generalization to articulated tools, a comprehensive evaluation is needed. (4) Currently, VLMGINEER is optimized for individual, isolated tasks, and we have not explored multitask optimization or generalization across diverse tasks. Future work should focus on validating designs with real-world robot experiments, enhancing action representations to include dynamic control, and exploring richer, articulated tool designs and multitask generalization.

References

[1] H. Shi, H. Xu, S. Clarke, Y. Li, and J. Wu, "Robocook: Long-horizon elasto-plastic object manipulation with diverse tools," 2023. [Online]. Available: https://arxiv.org/abs/2306.14447

[2] C. Qi, Y. Wu, L. Yu, H. Liu, B. Jiang, X. Lin, and D. Held, "Learning generalizable tool-use skills through trajectory generation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[3] A. Car, S. S. Yarlagadda, A. Bartsch, A. George, and A. B. Farimani, "Plato: Planning with llms and affordances for tool manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2409.11580

[4] K. Shaw, Y. Li, J. Yang, M. K. Srirama, R. Liu, H. Xiong, R. Mendonca, and D. Pathak, "Bimanual dexterity for complex tasks," in *8th Annual Conference on Robot Learning*, 2024.

[5] T. Chen, E. Cousineau, N. Kuppuswamy, and P. Agrawal, "Vegetable peeling: A case study in constrained dexterous manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2407.07884

[6] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. Gonzalez Arenas, H.-T. Lewis Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," *Arxiv preprint arXiv:2306.08647*, 2023.

[7] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.

[8] I. Huang, G. Yang, and L. Guibas, "Blenderalchemy: Editing 3d graphics with vision-language models," in *European Conference on Computer Vision*. Springer, 2024, pp. 297–314.

[9] L. Le, J. Xie, W. Liang, H.-J. Wang, Y. Yang, Y. J. Ma, K. Vedder, A. Krishna, D. Jayaraman, and E. Eaton, "Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model," *arXiv preprint arXiv:2410.13882*, 2024.

[10] A. Goldberg, K. Kondap, T. Qiu, Z. Ma, L. Fu, J. Kerr, H. Huang, K. Chen, K. Fang, and K. Goldberg, "Blox-net: Generative design-for-robot-assembly using vlm supervision, physics simulation, and a robot with reset," 2024. [Online]. Available: https://arxiv.org/abs/2409.17126

[11] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, 2024.

[12] V. Aglietti, I. Ktena, J. Schrouff, E. Sgouritsa, F. J. Ruiz, A. Malek, A. Bellot, and S. Chiappa, "Funbo: Discovering acquisition functions for bayesian optimization with

funsearch," *arXiv preprint arXiv:2406.04824*, 2024.

[13] AlphaEvolve team, "Alphaevolve: A gemini-powered coding agent for designing advanced algorithms," https://deepmind.google/discover/blog, May 2025, deepMind Blog.

[14] K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff, "Physical design using differentiable learned simulators," *arXiv preprint arXiv:2202.00728*, 2022.

[15] Y. Li, T. Kong, L. Li, Y. Li, and Y. Wu, "Learning to design and construct bridge without blueprint," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2398–2405.

[16] M. Guo, Z. Liu, S. Tian, Z. Xie, J. Wu, and C. K. Liu, "Learning to design 3d printable adaptations on everyday objects for robot manipulation," 2024.

[17] H. Zhanpeng and C. Matei, "Morph: Design co-optimization with reinforcement learning via a differentiable hardware model proxy," *IEEE Robotics and Automation*, 2024.

[18] Y. Wu, S. Kasewa, O. Groth, S. Salter, L. Sun, O. P. Jones, and I. Posner, "Imagine that! leveraging emergent affordances for 3d tool synthesis," *arXiv preprint arXiv:1909.13561*, 2019.

[19] H. Ha, S. Agrawal, and S. Song, "Fit2Form: 3D generative model for robot gripper form design," in *Conference on Robotic Learning (CoRL)*, 2020.

[20] X. Xu, H. Ha, and S. Song, "Dynamics-guided diffusion model for robot manipulator design," *arXiv preprint arXiv:2402.15038*, 2024.

[21] R. Liu, J. Liang, S. Sudhakar, H. Ha, C. Chi, S. Song, and C. Vondrick, "Paperbot: Learning to design real-world tools using paper," 2024.

[22] M. Li, R. Antonova, D. Sadigh, and J. Bohg, "Learning tool morphology for contact-rich manipulation tasks with differentiable simulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1859–1865.

[23] M. Jelisavcic, K. Glette, E. Haasdijk, and A. Eiben, "Lamarckian evolution of simulated modular robots," *Frontiers in Robotics and AI*, vol. 6, p. 9, 2019.

[24] D. J. Hejna III, P. Abbeel, and L. Pinto, "Task-agnostic morphology evolution," *arXiv preprint arXiv:2102.13100*, 2021.

[25] K. Walker and H. Hauser, "Evolution of morphology through sculpting in a voxel based robot," in *Artificial Life Conference Proceedings 33*, vol. 2021, no. 1. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 2021, p. 27.

[26] K. Sims, "Evolving virtual creatures," in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 2023, pp. 699–706.

[27] H. Dong, J. Zhang, T. Wang, and C. Zhang, "Symmetry-aware robot design with structured subgroups," 2023. [Online]. Available: https://arxiv.org/abs/2306.00036

[28] H. Dong, J. Zhang, and C. Zhang, "Leveraging hyperbolic embeddings for coarse-to-fine robot design," 2023. [Online]. Available: https://arxiv.org/abs/2311.00462

[29] I. Exarchos, K. Wang, B. H. Do, F. Stroppa, M. M. Coad, A. M. Okamura, and C. K. Liu, "Task-specific design optimization and fabrication for inflated-beam soft robots with growable discrete joints," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7145–7151.

[30] A. Zhao, J. Xu, M. Konaković-Luković, J. Hughes, A. Spielberg, D. Rus, and W. Matusik, "Robogrammar: graph grammar for terrain-optimized robot design," *ACM Trans. Graph.*, vol. 39, no. 6, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3414685.3417831

[31] A. Xie, F. Ebert, S. Levine, and C. Finn, "Improvisation through physical understanding: Using novel objects as tools with visual foresight," *arXiv preprint arXiv:1904.05538*, 2019.

[32] K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning," *Proceedings of the National Academy of Sciences*, vol. 117, no. 47, pp. 29 302–29 310, 2020.

[33] R. Girdhar, L. Gustafson, A. Adcock, and L. van der Maaten, "Forward prediction for physical reasoning," *arXiv preprint arXiv:2006.10734*, 2020.

[34] X. Lin, Z. Huang, Y. Li, J. B. Tenenbaum, D. Held, and C. Gan, "Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools," *arXiv preprint arXiv:2203.17275*, 2022.

[35] X. Lin, C. Qi, Y. Zhang, Z. Huang, K. Fragkiadaki, Y. Li, C. Gan, and D. Held, "Planning with spatial-temporal abstraction from point clouds for deformable object manipulation," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=tyxyBj2w4vw

[36] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese, "Learning task-oriented grasping for tool manipulation from simulated self-supervision," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 202–216, 2020.

[37] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese, "Keto: Learning keypoint representations for tool manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7278–7285.

[38] J. Brawer, M. Qin, and B. Scassellati, "A causal approach to tool affordance learning," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2020, pp. 8394–8399.

[39] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 6206–6213.

[40] Y. Noguchi, T. Matsushima, Y. Matsuo, and S. S. Gu, "Tool as embodiment for recursive manipulation," *arXiv*

preprint arXiv:2112.00359, 2021.

[41] M. Xu, P. Huang, W. Yu, S. Liu, X. Zhang, Y. Niu, T. Zhang, F. Xia, J. Tan, and D. Zhao, "Creative robot tool use with large language models," 2023.

[42] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," *Robotics: Science and systems foundation*, 2018.

[43] M. Toussaint, J.-S. Ha, and O. S. Oguz, "Co-optimizing robot, environment, and tool design via joint manipulation planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6600–6606.

[44] Y. Wang, S. Wu, H. Fu, Q. FU, T. Zhang, Y. Chang, and X. Wang, "Curriculum-based co-design of morphology and control of voxel-based soft robots," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=r9fX833CsuN

[45] Y. Wang, S. Wu, T. Zhang, Y. Chang, H. Fu, Q. Fu, and X. Wang, "Preco: Enhancing generalization in co-design of modular soft robots via brain-body pre-training," in *Conference on Robot Learning*. PMLR, 2023, pp. 478–498.

[46] K. S. Luck, H. B. Amor, and R. Calandra, "Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 854–869. [Online]. Available: https://proceedings.mlr.press/v100/luck20a.html

[47] Y. Yuan, Y. Song, Z. Luo, W. Sun, and K. Kitani, "Transform2act: Learning a transform-and-control policy for efficient agent design," *arXiv preprint arXiv:2110.03659*, 2021.

[48] A. Spielberg, A. Zhao, Y. Hu, T. Du, W. Matusik, and D. Rus, "Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[49] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "Scalable co-optimization of morphology and control in embodied machines," *Journal of The Royal Society Interface*, vol. 15, no. 143, p. 20170937, 2018.

[50] A. Mertan and N. Cheney, "Investigating premature convergence in co-optimization of morphology and control in evolved virtual soft robots," 2024. [Online]. Available: https://arxiv.org/abs/2402.09231

[51] R. P. Ringel, Z. S. Charlick, J. Liu, B. Xia, and B. Chen, "Text2robot: Evolutionary robot design from text descriptions," 2025. [Online]. Available: https://arxiv.org/abs/2406.19963

[52] T. Liao, G. Wang, B. Yang, R. Lee, K. Pister, S. Levine, and R. Calandra, "Data-efficient learning of morphology and controller for a microrobot," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2488–2494.

[53] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, "Jointly learning to construct and control agents using deep reinforcement learning," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 9798–9805.

[54] D. Ha, "Reinforcement learning for improving agent design," *Artificial life*, vol. 25, no. 4, pp. 352–365, 2019.

[55] J. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik, "Evolution gym: A large-scale benchmark for evolving soft robots," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2201–2214, 2021.

[56] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros, "Learning to control self-assembling morphologies: a study of generalization via modularity," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[57] T. Chen, Z. He, and M. Ciocarlie, "Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning," *arXiv preprint arXiv:2008.04460*, 2020.

[58] Z. Liu, S. Tian, M. Guo, C. K. Liu, and J. Wu, "Learning to design and use tools for robotic manipulation," 2023. [Online]. Available: https://arxiv.org/abs/2311.00754

[59] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, "An end-to-end differentiable framework for contact-aware robot design," in *Robotics: Science and Systems XVII*, ser. RSS2021. Robotics: Science and Systems Foundation, Jul. 2021. [Online]. Available: http://dx.doi.org/10.15607/RSS.2021.XVII.008

[60] K. Kawaharazuka, T. Ogawa, and C. Nabeshima, "Tool shape optimization through backpropagation of neural network," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 8387–8393.

[61] K. Qiu, K. Ciebiera, P. Fijałkowski, M. Cygan, and Ł. Kuciński, "Robomorph: Evolving robot morphology using large language models," *arXiv preprint arXiv:2407.08626*, 2024.

[62] C. Chen, X. Xu, X. Wang, and X. Huang, "Large language models as natural selector for embodied soft robot design," 2025. [Online]. Available: https://arxiv.org/abs/2503.02249

[63] J. Song, Y. Yang, H. Xiao, W. Peng, W. Yao, and F. Wang, "Laser: Towards diversified and generalizable robot design with large language models," in *The Thirteenth International Conference on Learning Representations*, 2025.

[64] W. B. Langdon and R. Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[65] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. Eiben, "Evolutionary robotics: what, why, and where to," *Frontiers in Robotics and AI*, vol. 2, p. 4, 2015.

[66] L. Nair, N. Shrivatsav, and S. Chernova, "Tool macgyvering: A novel framework for combining tool substitution and construction," *arXiv preprint arXiv:2008.10638*, 2020.

[67] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," 2018. [Online]. Available: https://arxiv.org/abs/1803.07635

[68] N. Di Palo and E. Johns, "Keypoint action tokens enable in-context imitation learning in robotics," in *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[69] Y. Yin, Z. Wang, Y. Sharma, D. Niu, T. Darrell, and R. Herzig, "In-context learning enables robot action prediction in llms," in *ICRA*, 2025.

[70] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, 2020.

[71] I. F. Jacobs, A. von Bayern, and M. Osvath, "A novel tool-use mode in animals: New caledonian crows insert tools to transport objects," *Anim. Cogn.*, vol. 19, no. 6, pp. 1249–1252, Nov. 2016.

[72] B. Ellenberger, "Pybullet gymperium," https://github.com/benelot/pybullet-gym, 2018–2019.

[73] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su, "SAPIEN: A simulated part-based interactive environment," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[74] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set," *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, pp. 36–52, Sep. 2015.

[75] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2019. [Online]. Available: https://arxiv.org/abs/1910.10897

VIII. Appendix

A. Baseline Details

1) Human-Prompted Designs Experiment Implementation: Each participant underwent the following experimental procedure for each task: (i) We provided a screenshot of the environment and a description of the task, accompanied by a brief Q&A session to ensure the participants understood the task. (ii) Participants then had five minutes to write a prompt in English specifying their desired tool design and robot action. We instructed participants to be as descriptive as possible while focusing on both the design of the tool and how the robot should use it to accomplish the task. (iii) we integrated their prompt into our standardized request to the VLM (by adding instructions as shown in Appendix VIII-D9), generating 5 tool described in URDF format along with a batch of 10 samples of action waypoints for each tool. (iv) The VLM outputs were then evaluated in our simulation environment using the same reward metrics described in Section 6. (v) Finally, we evaluated and recorded the best-performing tool and action pair based on the task reward metric for each participant. For a case study, we obtained prompts from three humans coming from three different backgrounds, including an LLM expert (a student with extensive research experience in LLM), a robotics expert (a student with extensive research experience in robotics), and a layperson (with no technical background). This case study will serve as an initial attempt on the concept. In the future, we plan to recruit more human subjects to conduct human study experiments on a larger sample population.

2) No-Tool Experiment Implementation: In the no-tool baseline experiment, we evaluate the robot's performance without any additional tool attachment. The Franka Panda robot uses its original two-finger gripper to perform the task, with the VLM generating action waypoints for the robot end effector pose and gripper open/close, totaling 7 degrees of freedom. The prompt for this baseline is adapted from our proposed prompt by removing the tool design component and associated instructions, while retaining the task description and action generation requirements. We use 5 agents with each generating 10 samples of action waypoints, evaluated using the same metrics introduced in Section 6. The complete no-tool prompt is provided in Appendix VIII-D8.

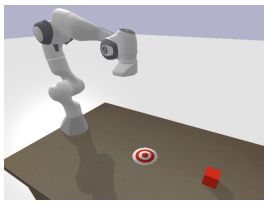
3) RLBench Experiment Implementation: In the RLBench experiment, we evaluate the robot's performance with tools from RLbench. We assume the tool is already attached to the end effector without considering the picking step. The tool are scaled to adapt to our tasks which are similar to the ones in RLBench. The prompt for this baseline is also adapted from our proposed prompt by removing the tool design component and associated instructions. We use 5 agents with each generating 10 samples of action waypoints, evaluated using the same metrics introduced in Section 6. The complete no-tool prompt is provided in Appendix VIII-D10.

B. RoboToolBench Details

Next, we provide detailed descriptions of each task and their corresponding dense reward functions in RoboToolBench.

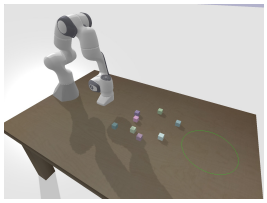
BringCube

In this task, a red cube on the desk which is out of the reach of the robot is needed to be brought closer to the target zone.
The reward measures how close the cube is to the target as a fraction of its starting distance, and scales it to 0~1.



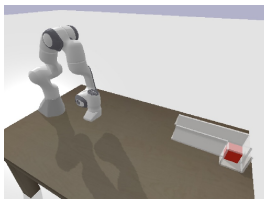
CleanTable

In this task, the colorful cubes representing dusts need to be pushed away from the robot into a circular target zone marked by the green boundary.
The reward reflects, on average, how far each cube has been pushed toward the goal circle, and scales it to 0~1.



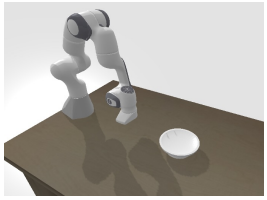
DislodgeCube

In this task, a red cube is confined within a white, transparent pipe in front of the robot, which has two exits: one opening faces the robot (along negative X) and the other at the front-right corner (along negative Y). The objective is to dislodge the cube through either opening.
The reward captures the cube's progress toward either of the two pipe exits by computing two separate, normalized (on a 0~1 scale) "distance-to-exit" scores and then taking the better one.



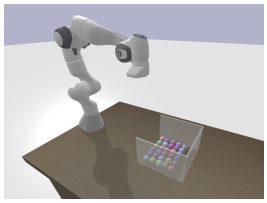
ElevatePlate

In this task, a white plate placed on the desk in front of the robot needs to be securely lifted up.
The reward measures how far the plate has moved from its starting position to the desired lifted position, and scales it to 0~1.



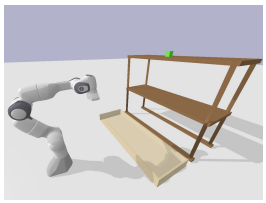
GatherSpheres

In this task, an open three-walled container filled with small purple spheres is placed before the robot. The objective is to gather and elevate as many spheres as possible above 0.3 m.
The reward captures, on average, how high the spheres have been lifted up to a specified cap, and scales it to 0~1.



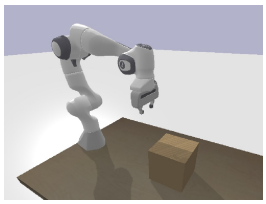
HighObject

In this task, a green cube sits on the top shelf. The objective is to place it inside the beige box positioned between the shelf and the robot.
The reward combines a hard "in-box" check with a smooth distance-based signal and a bonus for lowering the cube off the shelf, and scales it to 0~1.



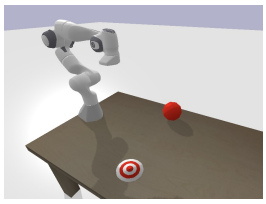
LiftBox

In this task, a brown box on the desk in front of the robot must be lifted above a height threshold of 0.25 m.
This reward measures how much the box has moved toward its target (lifted) position, and scales it to 0~1.



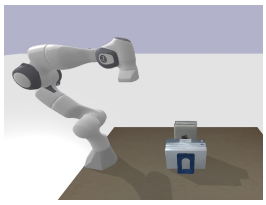
MoveBall

In this task, a red ball on the desk must be moved from the robot's left side to its right side.
The reward balances two objectives, getting the ball toward the right-side target and keeping its speed in check, and scales it to 0~1.



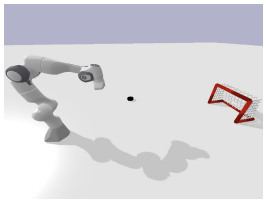
OneBook

In this task, two book holders with five books between them are in front of the robot. The objective is to pull out the middle (3rd) book while keeping the others in place.
This reward balances two goals, pulling out the middle book and keeping the others perfectly still, and scales it to 0~1.



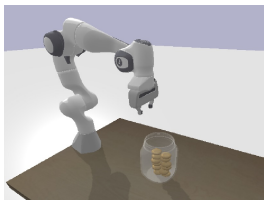
ScoreGoal

In this task, a hockey puck and a goal are placed on the ground far from the robot. The objective is to place the puck inside the goal.
The reward gives full credit once the puck is entirely inside the goal's 3D bounding box, and otherwise scales linearly with how much closer the puck is, horizontally, to the goal than it was at the start, and scales it to 0~1.



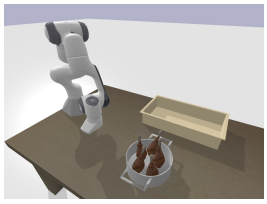
SnatchCookie

In this task, a transparent jar of cookies sits on the desk in front of the robot. The objective is to take at least one cookie from the jar.
The reward checks whether any cookie has been lifted out of the jar, and otherwise gives partial credit, from 0 to 1, based on how high the tallest cookie has been raised.



TurkeyLegs

In this task, a silver pot with handles on both sides, full of turkey legs, sits on the desk in front of the robot. To the pot's left (robot's perspective) is a chef's box. The objective is to transfer all turkey legs into the box without moving the pot.
The reward combines two checks, keeping the pot out of the box and getting each turkey leg into the box, by multiplying, and scales it to 0~1.



C. VLMGINEER *Implementation Details*

1) Design Agents Context: This section describes the context that applies to a single design agent. The design agent is provided with task context as illustrated in Fig. 2, which includes (1) the environment code, (2) a screenshot of the environment, (3) a brief task description, and (4) a total text prompt composed by the prompts in Appendix VIII-D. We also provide task-agnostic context, including (i) environment base class, which provides basic task-agnostic functionalities, (ii) environment runner, which establishes the context for how we will use the output tool and action, and (iii) a URDF of the Franka Panda without the tool to indicate where to attach the tool. All task environments are implemented as child classes of the environment base class, ensuring they inherit the fundamental functionalities while allowing for task-specific implementations.

2) Design Agents Queries: In VLMGINEER, we query VLM for designs by first initializing n_{agents} number of agents in parallel with the same prompts. For each agent, we prompt it to generate n_{tool} number of tool designs, and n_{action} number of action waypoint samples that correspond to *each* tool design. Therefore, the total number of tool-action pairs that are generated via one complete query is $n_{agents} \times n_{tool} \times n_{action}$. The prompt we use to specify this behavior to each agent is presented in Appendix VIII-D2.

We explicitly choose this style of querying to maximize time efficiency and design diversity: (1) time efficiency is achieved by reducing the querying algorithmic complexity by using parallel VLM agents. (2) Empirically, we found that design diversity is achieved when we balance dependence and independence between design decisions. Specifically, when a single VLM agent auto-regressively generates $n_{tool} \times n_{action}$ tool-actions pairs, having later design outputs be conditioned on previous design outputs can encourage diversity within that conditional distribution. However, in order to sample from many distinct conditional distributions, as this provides additional diversity, we found that parallel VLM queries that share no history can help with that. Ultimately, we found that optimizing time efficiency and design diversity led to better and faster initial samples as well as evolutions.

See Appendix VIII-C6 for details on the values we used for these parameters for benchmarking.

3) Design Agent Outputs: As a part of our prompt to the VLM to query for designs, we specified our desired tool and action formats. For our tool design requirements, please refer to Appendix VIII-D3 for details. For our action design requirements, please refer to Appendix VIII-D4 for details. Notably, these prompts are separated into with & without the Franka gripper usage. During VLMGINEER's initial sampling, some design agents are asked to design tools for the gripper, some are not. This ensures the full capabilities of the default morphology are used. The two types of tools are also specified with different required attachment locations: gripper-using tools are asked to be attached to the two Franka gripper fingers, and non-gripper-using tools are asked to be attached to a "virtual joint", which is a joint we set up positioned at the flange of the Franka end effector to make the attachment process more standardized.

4) Simulation Evaluation: From the previous section, we obtain a list of tool-action pairs in the form of URDF designs and action waypoints, respectively. To use these for simulation evaluation, we first merge the tool URDF without modification into a blank Franka Panda URDF (a blank Franka URDF will contain a gripper if gripper usage is enabled, and otherwise will not). For the action waypoints, which are inherently sparse, we implement linear interpolation for the position trajectory and SLERP (Spherical Linear Interpolation) for the orientation trajectory. The Pybullet simulation then executes these interpolated actions in the designated task environment. Finally, the environment returns result metrics for each run with the corresponding samples, allowing for both choosing evolution candidates and for producing the quantitative evaluation of the design performance. To speed up the evaluation, k_{sim} samples are evaluated in parallel Pybullet simulations at a time.

5) Evolution: After evaluating all previous tool-action pairs, we perform selection as follows: (1) For every task, we define two parameters to control the behavior of selection: $reward_{save}$ and k_{top}. (2) Using these parameters, we first select the k_{top} number of tool-action pairs with the highest task rewards, and then keep only the pairs that have a reward higher than the $reward_{save}$ threshold, resulting in a set of winner tool-action pairs. We found this selection mechanism empirically allows for the best signals for evolution.

We then take this winner tool-action pair set and feed it as context into the next design agent query. These previous designs are introduced to the VLM by the "evolution mission introduction prompt" in Appendix VIII-D1, where the VLM is asked to perform mutation and crossover on the previous tools via the rules specified in VIII-D7. These evolved design samples will be fed into the simulation for evaluation, and the

TABLE I: Benchmarking Parameters for Different Tasks

Task Name	n_{agent}	n_{tool}	n_{action}	k_{top}	$reward_{save}$	$n_{iteration}$	k_{sim}
BringCube	20	10	10	5	0.6	3	100
CleanTable	20	10	10	5	0.6	3	100
DislodgeCube	20	10	10	5	0.6	3	100
ElevatePlate	20	10	10	5	0.6	3	100
GatherSpheres	20	10	10	5	0.6	3	100
HighObject	20	10	10	5	0.5	3	100
LiftBox	30	15	15	5	0.1	3	100
MoveBall	20	10	10	5	0.6	3	100
OneBook	20	10	10	5	0.4	3	100
ScoreGoal	20	10	10	5	0.4	3	100
SnatchCookie	5	5	5	5	0.3	3	100
TurkeyLegs	30	10	15	5	0.2	4	100

cycle will continue. We define a final $n_{iteration}$ parameter to control the number of iterations that this cycle would go on for.

See Appendix VIII-C6 for details on the values we used for these parameters for benchmarking.

6) VLMGINEER *Benchmarking Details:* When benchmarking VLMGINEER against ROBOTOOLBENCH, we used a different set of parameters for each task, detailed in Table. I. We used `gemini-2.5-pro-preview-03-25` as our VLM model throughout the entire experiment, and ran PyBullet evaluations on an AMD Ryzen 7 9800X3D 8-Core Processor CPU with 64 GB of RAM. On average, one run of VLMGINEER on one of these tasks should take around 30 minutes.

D. Full Prompts

In this section, we provide all VLMGINEER prompts. We show individual prompt components in section VIII-D1-VIII-D7. We then describe we compose these prompts for different experiments in section VIII-D8-VIII-D10. For details of their usage, please refer to Appendix VIII-C.

1) Mission Introduction: Initial sampling mission introduction prompt:

```
You are a robotics hardware and controls
expert. You operate with boldness and
brilliance in the physical realm. You
work with a robot arm that sits in the
origin of your environment. You will be
presented with some robotic tasks, and
will be asked to design tools and actions
to complete the task. Your goal is not
to complete the task to perfection in one
fell swoop. Instead, your meta-goal is to
generate a wide range of differentiated
good solutions over time, where one of
them will inevitably succeed.
```

Evolution mission introduction prompt:

```
You are a robotics hardware and controls
expert. You operate with boldness and
brilliance in the physical realm. The goal
is to create tools and actions to complete
a given task. You will be given a list
of previously generated tool designs via
JSON with URDF. Your goal is to evolve the
tool designs via mutation and crossover,
and generate the new best actions for the
evolved tools. This will be done in a way
that is similar to genetic algorithms,
and will be specified in detail in the
"Evolutionary Process" section below.
```

2) Procedure Instruction:

```
The procedure you will follow:
1. Receive Environment Descriptions:
The user will provide some detailed
environment descriptions, robotic task
instructions, and an initial image of the
workspace area from the overhead camera.
2. Describe the Scene: Analyze the
environment. Write down the spatial
relationship, including by not limited to
the position, orientation, dimension, and
geometry of all the objects in the scene.
Use all the information provided to you,
including all text, code, and images.
3. Create Strategies and Designs: You
will need to create n_tool tool that you
can use to complete the task. For each of
the tools you designed, you must generate
n_action set of action waypoints that you can
use to complete the task. Specifically,
for a total of n_tool times, do the following
steps:
    (a) First, write down a completely
    different, out-of-the-box tool design
    to tackle the task. Make it unlike
    any other tool design you made in your
    other strategies.
    (b) Create these tools following the
    "Tool Specification" section below.
    (c) For this tool, write the following
    down: (1) The spatial relationship
    (pose transformation) between the
    end-effector and each component of the
    tool; (2) The 3D space that each tool
    component will take up when connected
    to the robot; (3) The usage of each
    component of the tool when carrying out
    the task.
```

```
(d) Use your previous analysis to tweak
any obvious issues with the position,
orientation, and dimension of your tool
design.
(e) Next, using your knowledge of
the tool and your in depth analysis
regarding the intricate 3D spatial
relationships between the tool and its
environment, create n_action number of
different step by step action plans
to enable to effective tool use (See
more in "Desired Action Criteria
Definitions"). Be very wary about how
objects interact with each other
(f) Transform your step-by-step action
plan into waypoints adhering to the
"Action Specifications". During
this transformation, think about
the inherent nature of controlling
robots with waypoint control and the
difficulty that may present.
```

3) Tool Specifications: Tool specification prompt without the use of Franka Grippers:

```
(Tool Specifications) Your design of the
tool must follow these rules: (1) You must
only use 3D rectangles for each component;
(2) Your tool will be outputted in a URDF
block format, which should be directly
added to the end of a panda URDF file,
before the robot closing declaration; (3)
Make sure your tools weigh very little in
the URDF file, where each tool part should
weigh no more than a few grams (these
weights do not have to be realistic, it
is just for the robot inverse kinematics
to have a easier time converging). (4)
Your design will be a single rigid tool,
which should be attached directly to the
"panda_virtual" link, which you can safely
assume to have the same orientation as
the world frame. (5) Any attachments you
design should geometrically be directly
connected to their parent links in the
URDF (there should be no gaps in between!)
(6) As a general observation, you perform
better when the tools you design are
complex and intricate.
```

Tool specification prompt with the use of Franka Grippers:

```
(Tool Specifications) Your design of the
tool must follow these rules: (1) You must
only use 3D rectangles for each component;
(2) Your tool will be outputted in a URDF
block format, which should be directly
added to the end of a panda URDF file,
before the robot closing declaration; (3)
Make sure your tools weigh very little in
the URDF file, where each tool part should
weigh no more than a few grams (these
weights do not have to be realistic, it
is just for the robot inverse kinematics
to have a easier time converging). (4)
Your design will be a pair of attachments
to the robot gripper fingers (which allows
the tool to be actuated with the robot
gripper); You should attach the left
attachment to "panda_leftfinger" and the
right attachment to "panda_rightfinger".
(5) Any attachments you design should
geometrically be directly connected to
their parent links in the URDF (there
should be no gaps in between!) (6) As a
general observation, you perform better
when the tools you design are complex and
intricate.
```

4) Action Specifications: Action specification prompt without the use of Franka Grippers:

```
(Action Specifications) Your tool-using
action will be a Nx6 numpy array of
action waypoints, where N is the number
of waypoints, and each waypoint is of
dimension 6 (xyz position + roll-pitch-yaw
euler angle orientations). Your action
needs to be precisely six numbers
per waypoint. Your waypoints will be
carried out by the EnvRunner class. It
is important to stress this: the action
waypoints are controlling the robot
end-effector "panda_virtual" link: this
means you have to carefully take into
account the dimensions of the tool and
the thickness of its parts when designing
effective waypoints. Again, you can
safely assume the end-effector has the
same orientation as the world frame upon
initialization (see frame clarification
again for details)!
```

Action specification prompt with the use of Franka Grippers:

```
(Action Specifications) Your tool-using
action will be a Nx7 numpy array of
action waypoints, where N is the number
of waypoints, and each waypoint is of
dimension 7 (xyz position + roll-pitch-yaw
euler angle orientations + binary gripper
open/close state in integers [0 for open,
1 for closed]). Your action needs to be
precisely seven numbers per waypoint.
Your waypoints will be carried out by the
EnvRunner class. It is important to stress
this: the action waypoints are controlling
the robot end-effector "panda_virtual"
link: this means you have to carefully
take into account the dimensions of the
tool and the thickness of its parts when
designing effective waypoints. Again, you
can safely assume the end-effector has the
same orientation as the world frame upon
initialization (see frame clarification
again for details)!
```

5) Action Diversity Specification:

```
(Desired Action Criteria Definitions)
For the description below, we will call
```

```
a single sequential set of waypoints in
a single rollout as one "action set".
For each tool you created, the goal
is to generate n_action action sets that
optimize the task success and motion
differentiation. Task success is optimized
when an action set is able to complete the
task successfully. Motion differentiation
is optimized when there exists a large
variance in the motion taken across all
action sets you design for the same tool.
A large variance in motion is defined the
tool, at each time step, is located at a
different location in the 3D space. Think
about how a tool can be used to interact
with the object from many different sides,
angles, and ways. When both conditions are
met, you have successfully designed a good
set of actions sets.
```

6) Frame Clarifications:

```
(Frame Clarification) In the world frame,
front/back is along the x axis, left/right
is along the y axis, and up/down is along
the z axis with the following directions:
Positive x: Towards the front of the table.
Negative x: Towards the back of the table.
Positive y: Towards the left. Negative
y: Towards the right. Positive z: Up,
towards the ceiling. Negative z: Down,
towards the floor. In terms of orientation,
starting from the origin frame, Positive
rotation about the x-axis: tilting the
end-effector head to the left. Negative
rotation about the x-axis: tilting the
end-effector head to the right. Positive
rotation about the y-axis: tilting the
end-effector head down. Negative rotation
about the y-axis: tilting the end-effector
head up. Positive rotation about the
z-axis: rotating the end-effector head
counter-clockwise. Negative rotation about
the z-axis: rotating the end-effector head
clockwise.
```

7) Evolutionary Instructions:

```
(Evolutionary Process) Your design
decision is a part of a tool design
genetic algorithm. For each of the n_tool
tool designs, you can choose to either
mutate or crossover. Specifically, tool
mutation is defined as one change to a
single randomly selected previous tool
design. Mutation changes include:
    (1) Changing the dimension, location,
    or orientation of a single component of
    the tool.
    (2) Adding, removing, or replacing a
    single component of the tool.
Crossover is defined as the process of
combining two randomly selected previous
tool designs to create a new tool design.
Combination is defined as:
    (1) Selecting components from two
    previous tool designs and combining
    them to form a new tool design.
All mutation and crossover decisions must
potentially increase the likelihood of
task success, yet all decisions must be
different and diverse.
```

8) No Tool Instructions:

```
You are a robotics hardware and controls
expert. You operate with boldness and
brilliance in the physical realm. You work
with a robot arm that sits in the origin
of your environment. You will be presented
with some robotic tasks, and will be asked
to design actions to complete the task.

...
```

The complete prompt is composed together with instructions from VIII-D2, VIII-D4, VIII-D5, and VIII-D6.

9) Human Specification Instructions:

```
You are a helpful robotics hardware and
controls expert. You have a robot arm that
sits in the origin of your environment.
You are working with a colleague as a
team to design tools and actions for a
robot to complete a task. Your colleague
will provide you with a design and action
instructions in the form of natural
language instructions. Your goal is to
use your colleague's design and action
instructions to output URDF and action
waypoints for the robot to use. You should
not use your own knowledge to design the
tool and action, but rather follow your
human colleague's instruction. Here is the
human colleague's prompt: {human_prompt}

...
```

The complete prompt is composed together with instructions from VIII-D2, VIII-D3, VIII-D4, VIII-D5, and VIII-D6.

10) RLBench Instructions:

```
You are a helpful robotics hardware and
controls expert. You have a robot arm that
sits in the origin of your environment.
You are working with a colleague as a
team to design tools and actions for a
robot to complete a task. Your colleague
will provide you with a design in the
format of a URDF, which is attached for
you as tool.txt. Your goal is to use your
colleague's URDF to come up with an action
plan for the robot to use.

...
```

The complete prompt is composed together with instructions from VIII-D2, VIII-D4, VIII-D5, and VIII-D6.

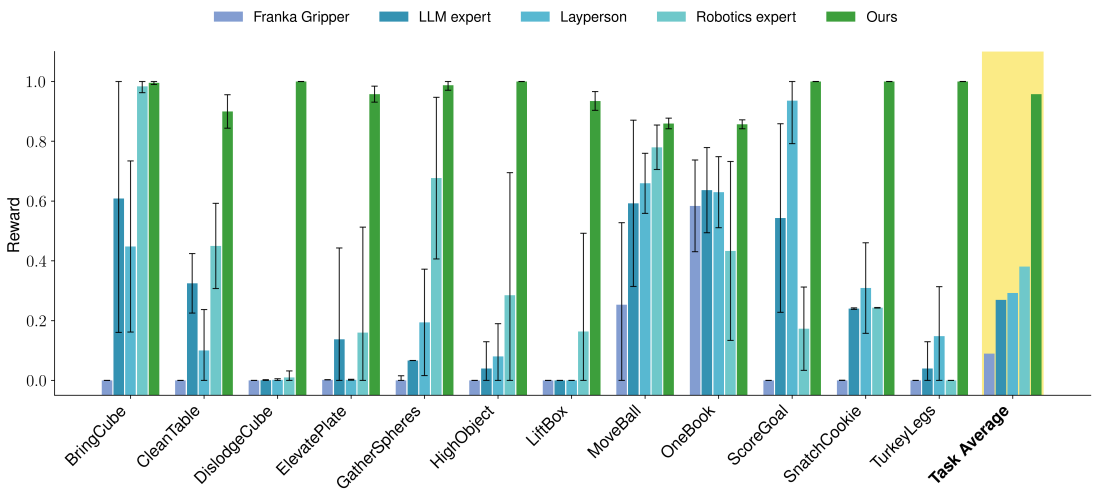


Fig. 8: Comparison of the mean and standard deviation of reward generated by VLMGINEER, human-prompted designs, and Franka Gripper across 12 tasks. Error bars represent standard deviation (clipped to the range $[0, 1]$).

E. Statistical Significance Analysis

Fig. 8 presents our primary quantitative results, including standard deviations across 5 runs. Across all 12 tasks, VLMGINEER consistently surpasses all baselines, exhibiting notably low variation between trials. This indicates that VLMGINEER reliably produces high-performing and stable tool-action designs.

In contrast, results obtained from human-prompted designs not only yield significantly lower performance but also show greater variations across runs. We attribute this discrepancy to several factors. First, human-specified tools often require more intricate control strategies; even if capable of completing the task, these designs tend to be less resilient to suboptimal or imperfect executions. By comparison, VLMGINEER-generated tools typically exhibit greater robustness to action imperfections. Second, human prompts sometimes suffer from specification ambiguity or misalignment with the VLM. There can be discrepancies between human intent and the VLM's internal representation and physical modeling capabilities. By automating the design process, VLMGINEER avoids these alignment issues, resulting in more effective and precisely realizable solutions.

F. Tool Design Gallery

Table II and III show our tool gallery. In this tool design gallery, we take the opportunity to display tools from a few tasks that seemed to have allowed VLMGINEER the most creative freedom. These are tool designs that are not presented elsewhere in the paper. We believe this illustrates VLMGINEER's impressive physical creativity and problem-solving capabilities.

G. Licenses

The cardboard box asset used in LIFTBOX environment is from PartNet-Mobility Dataset [73]. Their terms of use are stated here: sapien.ucsd.edu/about.

The book assets and the book holder used in the ONEBOOK environment came from the YCB Dataset [74]. This dataset is under the CC BY 4.0 license.

The goal frame and net assets used in the SCOREGOAL environment came from the Meta-World Benchmark [75]. This benchmark is under the MIT License.

The transparent jar asset used in the SNATCHCOOKIE environment came from cgtrader, a 3D CAD model website. This asset is under the "Royalty Free No Ai License", detailed here.

The cookie assets used in the SNATCHCOOKIE environment came from sketchfab, a 3D CAD model website. This asset is under the CC BY 4.0 license.

The turkey leg assets used in the TURKEYLEGS environment came from sketchfab, a 3D CAD model website. This asset is under the CC BY 4.0 license.

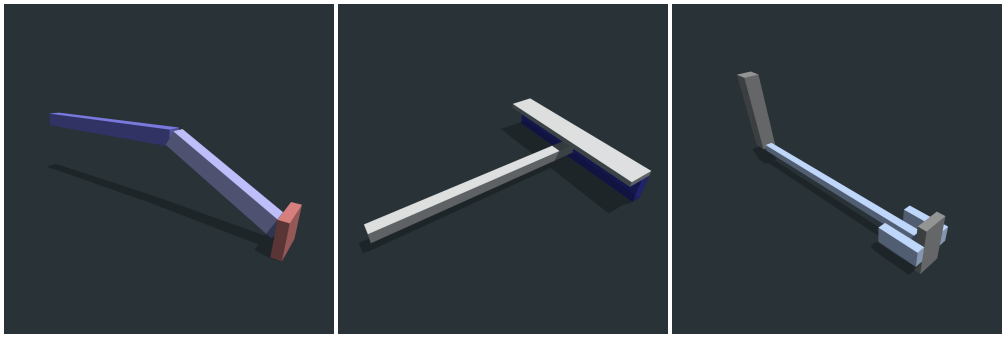
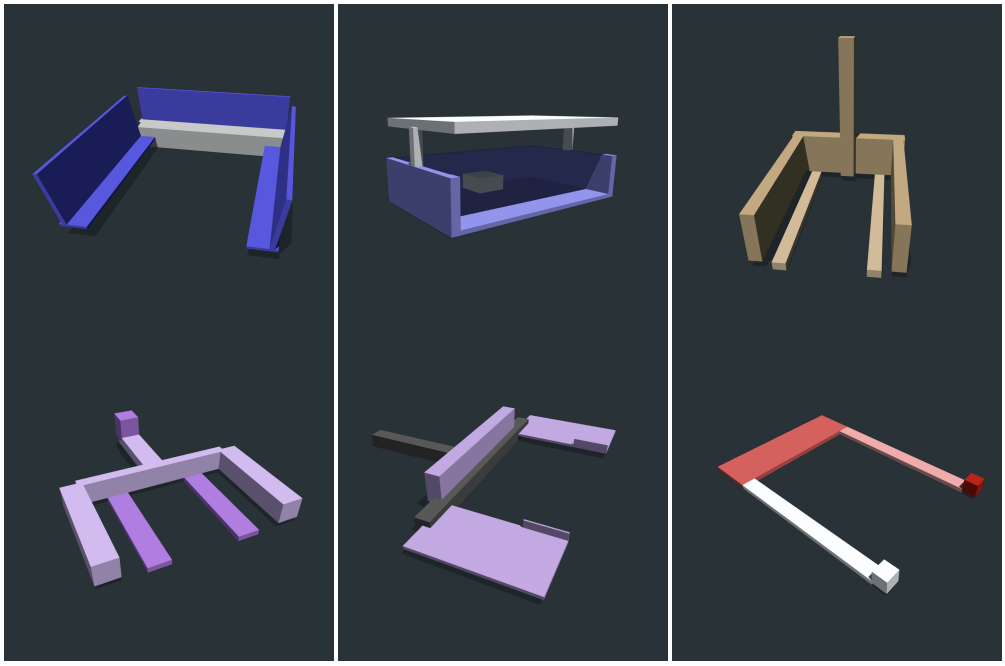
Task Name	Example Tool Designs
BRINGCUBE	
CLEANTABLE	
ELEVATEPLATE	

TABLE II: Tool-gallery for BringCube, CleanTable, and ElevatePlate.

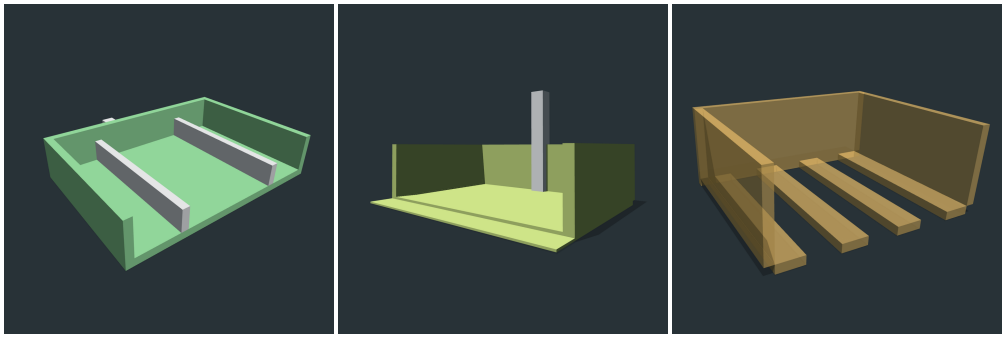
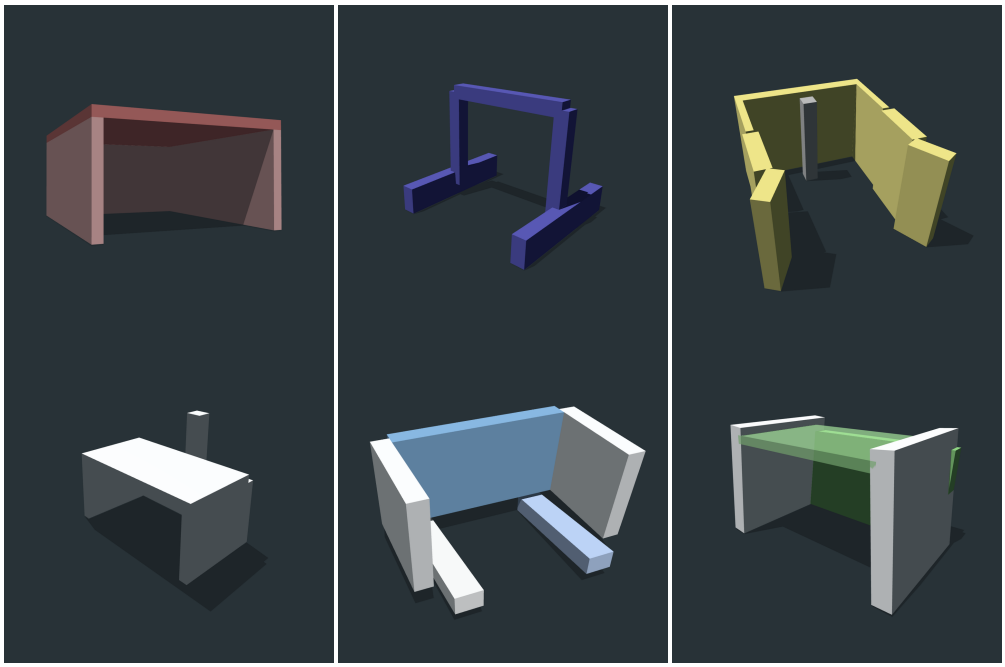
Task Name	Example Tool Designs
GATHERSPHERES	
MOVEBALL	

TABLE III: Tool-gallery for GatherSpheres and MoveBall.