RoboMorph: Evolving Robot Morphology using Large Language Models

Kevin Qiu[1,2*], **Władysław Pałucki**[1], **Krzysztof Ciebiera**[1], **Paweł Fijałkowski**[1], **Marek Cygan**[1,3], **Łukasz Kuciński**[1,2,4 *]
[1]University of Warsaw [2]IDEAS NCBR [3]Nomagic [4]Polish Academy of Sciences
kevinxqiu@gmail.com

Abstract

We introduce RoboMorph, an automated approach for generating and optimizing modular robot designs using large language models (LLMs) and evolutionary algorithms. In this framework, we represent each robot design as a grammar and leverage the capabilities of LLMs to navigate the extensive robot design space, which is traditionally time-consuming and computationally demanding. By introducing a best-shot prompting technique and a reinforcement learning-based control algorithm, RoboMorph iteratively improves robot designs through feedback loops. Experimental results demonstrate that RoboMorph successfully generates nontrivial robots optimized for different terrains while showcasing improvements in robot morphology over successive evolutions. Our approach highlights the potential of using LLMs for data-driven, modular robot design, providing a promising methodology that can be extended to other domains with similar design frameworks.

1 Introduction

Generative methods, such as large language models (LLMs), have increasingly influenced various domains of machine learning and everyday life, including robotics. LLMs are utilized to generate robot policies as code Liang et al. (2023), provide knowledge for executing complex instructions Ahn et al. (2022), enhance generalization and semantic reasoning Brohan et al. (2023b;a), design reward functions and domain randomization Ma et al. (2024a;b), and develop general policies for robot manipulation Team et al. (2024).

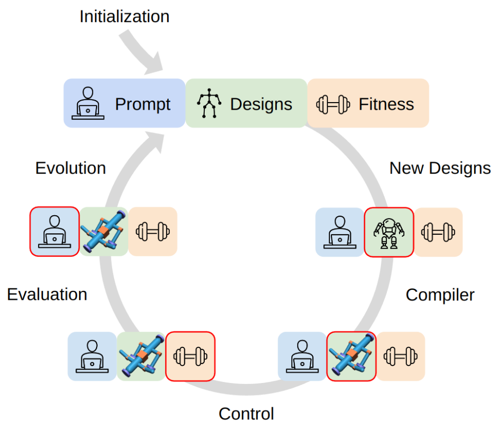


Figure 1: Visualization of one iteration of RoboMorph. At each stage, the prompt (blue), robot design (green), and fitness score (orange) are displayed. The element modified in each step is highlighted with a red border.

In this paper, we address the challenge of robot design, a fundamental problem in modern robotics Zeng et al. (2023). Traditional engineering approaches are often time-consuming and heavily depend on human designers to manually prototype, test, and iterate through an extended design cycle. This challenge is further compounded by the vast robot design space that must be explored. By automating key aspects of the design process, we can significantly reduce the time and costs associated with development. Furthermore, leveraging data-driven approaches has the potential to generate designs that surpass those conceived and produced by human designers.

Previous attempts to automate the design process have been slow and require searching through a vast design space, which is often computationally inefficient. One class of such approaches is

[*]Corresponding author.

inspired by evolutionary algorithms (EAs) found in nature Bäck & Schwefel (1993). However, EAs necessitate iterating through hundreds of generations, and existing solutions struggle to scale with increasing design complexity Thierens (1999).

In this work, we propose a method to address some of these challenges. We introduce `RoboMorph`, a novel approach to robot design that is data-driven, generative, and modular. `RoboMorph` draws inspiration from recent advancements in large language models (LLMs) OpenAI et al. (2024), prompting techniques Brown et al. (2020); Wei et al. (2022b), and compositional design generation based on a structured robot grammar Zhao et al. (2020). Additionally, we employ reinforcement learning (RL)-based control algorithms to compute fitness scores for each automatically generated robot design. An overview of our approach is presented in Figure 1.

We leverage LLMs in our method for several reasons. First, LLMs possess extensive prior knowledge and demonstrate emergent capabilities Wei et al. (2022a), which can fundamentally transform the methodology of robot design. Second, utilizing natural language as an input module enables a flexible approach that accommodates custom designs based on specific user requirements. For example, this flexibility allows users to specify particular designs tailored for specific tasks or environments. Finally, unlike search engines, LLMs can suggest ways to integrate knowledge into prompts and apply it to novel problems, thereby ensuring the generated designs are practical and effective.

We experimentally demonstrate that an iterative approach provides feedback to the LLM, enabling it to generate progressively improved designs over time. `RoboMorph` illustrates the feasibility of using LLMs for robot design across different environments, and we believe this approach can be extended to other domains that follow similar design principles. Our work represents a step toward a future in which robots are designed in a manner which can facilitate their rapid deployment in real-world applications.

Our contributions are as follows:

- We introduce `RoboMorph`, which, to the best of our knowledge, is the first approach to integrate LLMs, evolutionary algorithms, robot grammars, and RL-based control to automate robot design for a given task.
- Our modular framework enables greater customization than existing methods, allowing users to modify the pipeline to suit specific needs. For example, users can adjust the input prompt or modify the simulation environment.
- We present experimental results demonstrating the potential of `RoboMorph`, showcasing its effectiveness and benchmarking its improved designs against existing methods.

2 Problem Formulation

We formulate our problem with the objective of optimizing the morphology of a robot across different terrains. Below, we provide a detailed description of both key components.

2.1 Design Optimization

In episodic environments, RL models problems as a Markov Decision Process (MDP), defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ represents the transition dynamics, R is the reward function, and γ is the discount factor. The agent follows a policy $\pi(a_t|s_t)$, selecting actions $a_t \in \mathcal{A}$ based on the current state $s_t \in \mathcal{S}$. Starting from the initial state s_0, actions are sampled from π, leading to state transitions governed by $\mathcal{T}(s_{t+1}|s_t, a_t)$ with an associated reward r_t. The goal of RL is to determine the optimal policy π^* that maximizes the expected discounted reward: $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{H} \gamma^t r_t\right]$, where H denotes the time horizon.

An agent's design $D \in \mathcal{D}$ plays a crucial role in determining its functionality. In our framework, D encompasses both the agent's skeletal structure and component-specific attributes, such as limb lengths, joint types, and end effectors. To account for design variations, the original reward function $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{H} \gamma^t r_t\right]$ is extended to incorporate design dependence: $J(\pi, D) = \mathbb{E}_{\pi,D} \left[\sum_{t=0}^{H} \gamma^t r_t\right]$. Thus, the objective is to determine the optimal design D^* that maximizes the expected cumulative rewards:

$$D^* = \arg\max_D \max_\pi J(\pi, D). \tag{1}$$

To compute Equation 1, we must design the robot's morphology and train its control network. However, defining the design space $\mathcal{D}$ is challenging due to the vast number of possible skeletal structure combinations. Consequently, the training process must refine both the robot's morphology and its control network simultaneously. To simplify this co-optimization problem, we divide the overall process into two main stages and alternate between the *design stage* and the *control stage*.

In the design stage, the design D_{t+1} is updated based on the input prompt, which consists of the *system prompt*, *user prompt*, and *few-shot examples*, formally defined as:

$$D_{t+1} = f_{\texttt{LLM}}(\texttt{prompt}_{\texttt{input}}). \tag{2}$$

The mapping function $f_{\texttt{LLM}}$ determines which components of the robot should be modified or reconstructed. We hypothesize that the LLM can acquire this capability. To obtain a new policy π_{t+1}, we use RL to train a set of parameters θ from scratch, optimizing them to maximize: $J(\pi_{t+1}, D_{t+1}) = J(\pi_\theta, D_{t+1})$.

2.2 Terrains

Terrains define the tasks for which robot morphologies are optimized. Each terrain is designed to produce a distinct set of optimal designs. We extend the set of environments introduced in Robogrammar Zhao et al. (2020). Figure 5 provides visualizations of each terrain.

Ridged terrain. A series of evenly spaced ridges, averaging two meters apart, spans the entire width of this terrain. It requires locomotion designs capable of jumping or crawling for effective navigation.

Flat terrain. A featureless surface with a high friction coefficient of 1.0, accommodating a wide range of locomotion styles and designs.

Frozen terrain. A flat surface with a low friction coefficient of 0.05. This terrain challenges designs to either maximize traction or exploit the reduced friction for movement.

Beams terrain. A series of platforms with overhanging beams suspended in the air. Designs must navigate beneath the beams, favoring those that remain close to the ground while avoiding tall or upright postures.

3 RoboMorph

The framework introduced in this paper, `RoboMorph`, is illustrated in Figure 2. It follows an evolutionary pipeline that maintains a population of robot designs generated using an LLM (specifically, GPT-4o). At each iteration, the LLM prompt is provided with a series of few-shot examples, which serve as a guide for the LLM to reason about its design output (Sections 3.1 and 3.2).

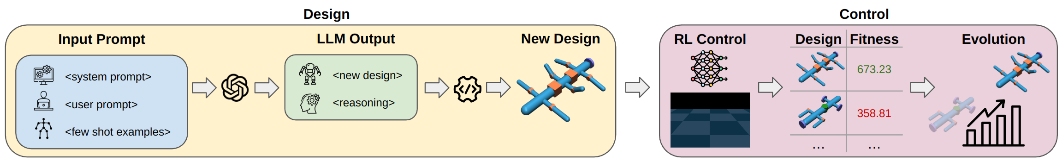


Figure 2: Overview of the `RoboMorph` framework, which consists of a *design stage* and a *control stage*. The process is divided into six steps: (1) The input consists of the system prompt, user prompt, and few-shot examples. (2) Based on the input prompt, the LLM generates a new design and provides its reasoning. (3) A compiler converts the LLM-generated design into an XML file, which can be rendered in a simulator. (4) An RL control policy is trained in simulation for the robot in a given environment. (5) The robot is evaluated, and a fitness score is assigned to each design. (6) The population of robots is updated through evolution by pruning the worst-performing candidates.

The designs are expressed as text that adheres to the rules of the robot grammar Zhao et al. (2020). This is particularly beneficial, as recursion and branching enable the generation of a diverse set of designs that are feasible for real-world construction. A compiler converts each new design into an XML file (Section 3.3), which is then fed into the MuJoCo Todorov et al. (2012) physics simulator (Section 3.4) to learn a control policy and compute a fitness score for each design (Section 3.5).

The new batch of designs is added to the existing population, and the worst candidates are pruned, retaining only the top-k best designs (Section 3.6). These top-k designs are subsequently incorporated into the few-shot examples for the next iteration. The entire pipeline is initialized with an initial set of few-shot robot design examples (Appendix A). In the following subsections, we detail each module and provide the pseudocode for `RoboMorph` in Algorithm 1.

3.1 Prompt Structure

We generate robot designs using the chat API of GPT-4o with its default settings and hyperparameters. The input to the model consists of three main components:

System prompt. This defines both the structural and component rules governed by the robot grammar Zhao et al. (2020), ensuring that the generated designs are physically feasible. The LLM is instructed to provide reasoning and a step-by-step explanation of how the structural rules are applied during the assembly process.

User prompt. This instructs the LLM to design a robot while explaining its reasoning in a step-by-step manner, inspired by the approach in Wei et al. (2022b). The user prompt remains fixed and is defined as: `"Your task is to design a single robot. Use the examples provided to guide your reasoning and explain your design step by step."`

Few-shot examples. This set consists of robot designs that have achieved the highest fitness scores up to the given evolution step. By including these examples, the LLM can iterate on existing designs and generate improved variations. For each example, we also provide the reasoning that was used in constructing the design. Further details are provided in Section 3.2.

Algorithm 1 Pseudocode for `RoboMorph`

```
Requires:
$\texttt{prompt}_{\texttt{system}}$        robot grammar rules
$\texttt{prompt}_{\texttt{user}}$          instructions for designing a robot
$\texttt{evolutions}$                      number of evolutions
$\texttt{population}$                      number of designs per population
$K$                                        number of few-shot examples
$\mathcal{Q}$ min-queue $\{(\texttt{fitness}_i, \texttt{design}_i, \texttt{reason}_i)\}_{i=1}^{K}$

function RoboMorph
fewshots = init_few_shots() {Appendix A}
for $i = 1$ to evolutions do
  for $j = 1$ to population do
    $\texttt{design}_{\texttt{new}}, \texttt{reason}_{\texttt{new}} \leftarrow LLM(\texttt{prompt}_{\texttt{system}}, \texttt{prompt}_{\texttt{user}}, \texttt{fewshots})$
    $\texttt{design}_{\texttt{xml}} \leftarrow Compiler(\texttt{design}_{\texttt{new}})$
    if $\texttt{design}_{\texttt{xml}}$ is corrupted then continue
    $\texttt{fitness}_{\texttt{new}} = Evaluate(\texttt{design}_{\texttt{xml}})$
    $\gamma_{\min} \leftarrow \arg\min_i \texttt{fitness}_i$ in $\mathcal{Q}$
    if $\texttt{fitness}_{\texttt{new}} > \mathcal{Q}[\gamma_{\min}].\texttt{fitness}$ then
      $\mathcal{Q}[\gamma_{\min}] \leftarrow (\texttt{fitness}_{\texttt{new}}, \texttt{design}_{\texttt{new}}, \texttt{reason}_{\texttt{new}})$
    end if
  end for
  fewshots $\leftarrow \emptyset$
  for all $(\texttt{fitness}, \texttt{design}, \texttt{reason}) \in \mathcal{Q}$ do
    fewshots.append(str(design, reason))
  end for
end for
end function
```

For a detailed listing of the prompts, see Appendix B.

3.2 Best-Shot Prompting

Inspired by the work in Brown et al. (2020), our motivation for appending few-shot examples in the prompt is threefold.

First, when initially testing our pipeline in a zero-shot manner, we observed hallucinations in the LLM's output, where generated designs were not plausible under the rules of robot grammar. This issue is common when using LLMs for creative tasks Romera-Paredes et al. (2024) and is particularly problematic, as it prevents us from compiling the LLM's textual output into a realizable robot that can be rendered in simulation. We further describe this issue in Section 3.3. To mitigate this, we include step-by-step robot-building examples as few-shot prompts to better ground the LLM in following the robot grammar rules.

Beyond improving the LLM's adherence to grammar constraints, the few-shot examples also serve to guide its design process more effectively. We frequently observed that the LLM could become trapped in a local minimum of creative designs, limiting its ability to generate diverse robot morphologies. Here, we define diversity as a qualitative measure based on the number of limbs and the variety of joints present in a robot's structure. To address this, we initialize the few-shot examples by randomly sampling robot grammar rules to construct a robot. The exact details of the few-shot initialization are outlined in Appendix A.

Finally, our results in Section 4.2 demonstrate that continuously updating the few-shot examples in the prompt with better robot designs—those with higher fitness scores—leads to progressively more optimal designs generated by the LLM over time. We refer to this process of updating the few-shot examples as *best-shot prompting* and further discuss this evolutionary approach in Section 3.6.

Notably, we do not provide the fitness values associated with each design, as we observed that doing so causes the LLM to overfit to those designs, thereby reducing exploration. Figure 3 presents examples of few-shot robot designs that were initialized at the beginning of the framework. The random generation of these morphologies introduces diverse and interesting designs, albeit ones that are initially unoptimized for forward locomotion.

3.3 Robot Design

The textual representation of a robot design, adhering to the rules of the robot grammar, is generated by the underlying LLM. Unlike the search algorithm proposed in Robogrammar, our method avoids the computational bottleneck of iteratively exploring multiple branches within a tree-like structure to identify the optimal design. Instead, the LLM directly generates a robot design, which is then compiled into an XML file compatible with the MuJoCo simulator.

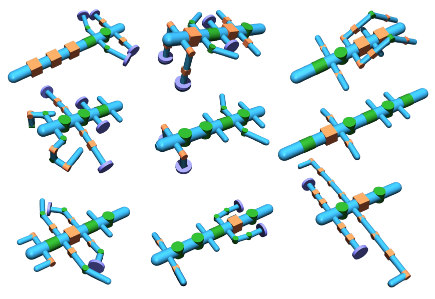


Figure 3: Examples of initial few-shot robot designs generated using Algorithm 2 (see Appendix A).

Although it is possible to convert the LLM's output directly into an XML file, our initial results indicated that GPT-4 was unreliable in performing this task without errors, a limitation also noted in Makatura et al. (2023). One important consideration is that we disable collisions between bodies in the robot XML to accelerate training when learning the corresponding control policy. While this may affect the absolute fitness values obtained, applying this modification consistently across all evaluated robots ensures that relative performance comparisons remain valid. In the final evaluation, we re-enable collisions for the best-performing design, as discussed in Section 4.3.

3.4 Simulation

Since our method relies on generating a large number of designs, each of which must be trained using an RL policy, it was crucial to optimize the training and evaluation loop for speed. To achieve this, we adopted MuJoCo XLA (MJX), a JAX-based implementation of the well-known MuJoCo physics simulator, capable of running on accelerated hardware. MJX leverages modern GPUs to enable parallelized experience collection across thousands of environment instances. Additionally, we utilized Brax Freeman et al. (2021), a JAX-based framework for RL, which seamlessly integrates with the MJX physics pipeline and provides implementations of popular RL algorithms such as Soft Actor-Critic (SAC) Haarnoja et al. (2018b) and Proximal Policy Optimization (PPO) Schulman et al. (2017). To further accelerate computation, we parallelized RL training, allowing multiple robot designs to be trained and evaluated simultaneously, each on a separate GPU. This parallelization, combined with the computational speedup from MJX, significantly increased both the number of evolutionary iterations and the number of designs generated per evolution.

3.5 Control

After compiling the design into XML format, each robot design is trained in simulation using the SAC algorithm, and the resulting policy is evaluated to compute a fitness score. We also experimented with PPO, but SAC was found to be more consistent in learning to control a diverse range of robot designs. Each design was evaluated using the same set of hyperparameters and the same number of environment steps (see Appendix C for details).

3.6 Evolution

Exploration. We consider the LLM as an evolution of search engines, capable of generating the ”most probable” response to a given prompt Shanahan (2024). However, LLMs are prone to hallucinations, which can result in outputs that appear plausible but are factually incorrect Bang et al. (2023). We leverage this phenomenon by treating the LLM as a generator of robot designs, with the primary goal of providing creative solutions. To facilitate this, we repeatedly prompt the LLM to generate a diverse population of robots, encouraging exploration while using a verifier to ensure the viability of the suggested designs.

Exploitation. Once a population of robot designs has been generated, we evaluate each design to filter out poorly suggested ideas from the exploration phase. As previously discussed, each robot is tested in simulation under a control policy and assigned a fitness score. Following an evolutionary approach, we retain and store the top-k designs with the highest fitness scores in a database while discarding the remaining candidates.

4 Experiments

4.1 Experimental Setup

We evaluate `RoboMorph` using 8 random seeds over 50 evolutionary generations, with a population size of 32 designs per evolution. Given the stochastic nature of the LLM’s output, our initial grid search for optimal evolution and population size proved ineffective, as different combinations yielded varying results across different seeds. As a result, we selected suitable values for these two hyperparameters based on similar work found in Fernando et al. (2023); Ringel et al. (2024). In total, 1,600 robot designs are evaluated per seed of the framework. During RL policy training, we evaluate 8 robot designs simultaneously in a cluster equipped with 8 NVIDIA A100-40GB GPUs. The total runtime of `RoboMorph` for a single seed amounts to approximately 200 GPU hours.

We assess the fitness of each design within a custom MJX environment and use the Brax implementation of the SAC algorithm. The state space is represented by a vector encoding the position and velocity of each joint, while the action space consists of joint actuations with elements ranging within $(-1, 1)$. The reward function is defined in Equation 3 as follows:

$$r_t(s, a, s') = \vec{v_x}(s, a, s'), \tag{3}$$

where $\vec{v_x}(s, a, s')$ represents the instantaneous forward velocity of the robot as it transitions from state s to s' using action a.

The fitness score of a design, shown in Equation 4, is defined as the average reward obtained by the corresponding policy learned in the control module. It is estimated using a Monte Carlo approach over $N = 128$ random rollouts:

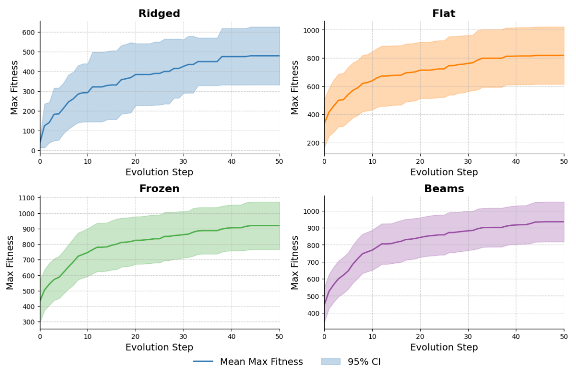
$$F(\texttt{design}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{j=1}^{T_i} r_{ij}, \tag{4}$$

where T_i denotes the length of the i-th rollout, and r_{ij} is the reward obtained at the j-th step of the i-th rollout. Rollouts terminate either when the agent becomes unhealthy or when the end of the episode is reached. The agent’s health is determined by checking whether its z-coordinate remains within a predefined range.

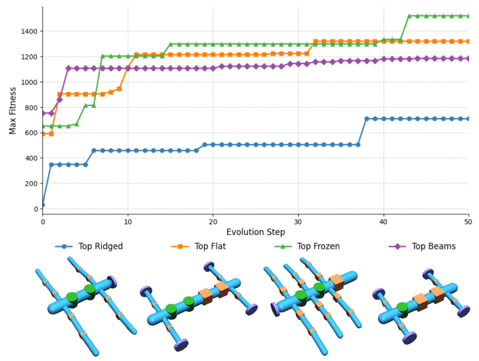
The architecture of both the critic and policy networks consists of fully connected layers of size [256, 256]. We employ the Adam optimizer Kingma & Ba (2017) and use ReLU activation functions across all layers in each network. The training process involves 1,000,000 environment interactions with a batch size of 256, striking a balance between computational efficiency and training stability. All other RL-related hyperparameters are listed in Appendix C, Table 1. To ensure a fair fitness assessment of each design, these hyperparameters remain constant throughout all experiments.

4.2 Main Experiment

In this section, we present the results obtained from RoboMorph. Figure 4(a) illustrates the average fitness score and the 95% confidence intervals across 8 seeds for the best-performing robot design in the population at each evolutionary step. A positive trend indicates that each iteration of RoboMorph functions as a robot design improvement operator. The confidence intervals highlight the stochastic nature of our approach, with two primary sources of randomness: the RL optimization algorithm and the evolutionary mechanism. Figure 4(b) depicts the evolution of the best-performing seed and the highest-scoring robot design obtained across all seeds for each terrain using RoboMorph. This represents the exploitation phase of our evolutionary algorithm (see Section 3.6).



(a) Average maximum fitness with 95% confidence intervals across 8 seeds for the best robot design in the population at each evolutionary step.



(b) Evolution of the best-performing seed for each terrain.

Figure 4: Maximum fitness of the best robot design in the population at each evolutionary step.

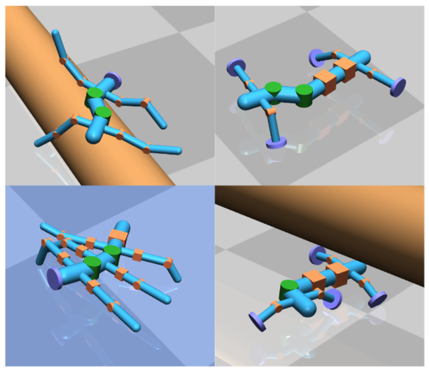


Figure 5: Best-performing robot designs generated by RoboMorph for each terrain: ridged (top-left), flat (top-right), frozen (bottom-left), and beams (bottom-right).

The best robot designs generated by RoboMorph across all seeds for each terrain are shown in Figure 5. As observed, the optimal design for the ridged terrain is characterized by long limbs with knee joints that can swing upwards to clear obstacles. The design for the flat terrain features shorter limbs with unactuated wheels spaced far apart along the body, enabling a full range of motion. The low inertia of short limbs, combined with free-spinning wheels, provides efficient and fast locomotion on obstacle-free terrains.

The frozen terrain requires a different strategy to overcome its frictionless surface. The design generated by RoboMorph features a hexapod with rigid joints positioned at the upper legs of all limbs, which allows the robot to maintain a wide base and a low center of gravity—both crucial for balance. Interestingly, the beam terrain produces a design similar to that of the flat terrain. This similarity is expected, given that both surfaces are identical. However, the design for the beam terrain has even shorter limbs, allowing the robot to traverse underneath the overhanging beams.

4.3 RoboMorph vs. Robogrammar

In this section, we benchmark `RoboMorph` against the best designs obtained from the original Robogrammar work. Unlike our method, which leverages LLMs to generate designs, Robogrammar introduces a Graph Heuristic Search algorithm that explores combinatorial design spaces. For a detailed explanation of their approach, we refer readers to the work in Zhao et al. (2020). The best designs obtained using Robogrammar are showcased in Appendix D.

To compare both methods, we first reconstruct the exact designs from Robogrammar within our environment, which consists of ridged, flat, and frozen terrains. For a fairer comparison, we enable collisions between bodies in our method—previously disabled to accelerate computation, as discussed in Section 3.3. This modification results in a variation in absolute fitness scores compared to the results presented in Figure 4(b). We then evaluate each method's proposed designs on their respective terrains across 8 seeds, training our RL control policy on each design. Finally, we present the average fitness scores along with error bars for both methods in Figure 6.

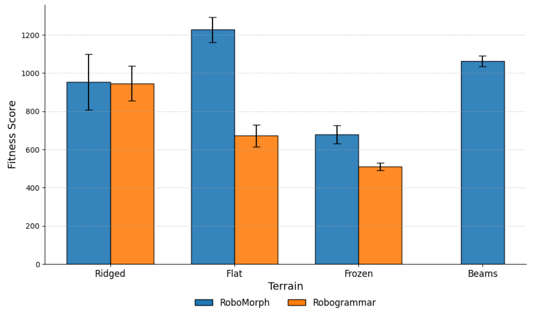


Figure 6: Average fitness of the best designs generated by `RoboMorph` and Robogrammar across 8 seeds.

Our comparison highlights several key observations regarding the best designs for each terrain between the two methods. First, while we present results for the beam terrain, we do not evaluate any Robogrammar designs in this environment, as this terrain is specific to our method. Second, the fitness scores obtained for the ridged terrain using both methods are similar, with overlapping error bars. This result is unsurprising given the morphological similarities between the two designs. Both designs feature a quadrupedal structure with two roll joints in the body and similar types and quantities of joints in the limbs.

The most notable performance difference between the two methods is observed on the flat terrain. While the arrangement of joints in the body could be subjectively debated between the two designs, the most striking distinction is the use of wheels as limb end effectors in the design generated by `RoboMorph`. The incorporation of wheels on flat terrain is advantageous, as continuous contact with the ground minimizes energy loss, enabling faster locomotion. In contrast, limb-based locomotion is inherently limited by stride length and frequency. Finally, on the frozen terrain, the design generated by `RoboMorph` also demonstrates an improvement over the design obtained using Robogrammar.

Robogrammar's frozen terrain design emphasizes the use of ”highly articulated yet compact arms to maintain contact with the ground during the stance phase, while the rear body segment slides freely.” Similar reasoning can be applied to the design generated by `RoboMorph`. We observe the use of rigid joints in the upper legs of each limb, extending the reach of each stride while maintaining a low center of gravity—crucial for locomotion on low-friction surfaces. Additionally, the rear rigid joint in the body extends the tail, similarly allowing the rear body segment to slide freely. We hypothesize that the hexapod structure of the design from `RoboMorph` provides enhanced stability and more efficient forward motion compared to the bipedal structure presented in Robogrammar.

While the results of `RoboMorph` benchmarked against Robogrammar appear promising, several important considerations regarding our comparison should be noted. First, we manually reconstructed Robogrammar's designs using XML bodies to the best of our ability. Although the overall morphology remains identical, there may be minor discrepancies in physical parameters such as mass and size. Second, it is important to highlight that Robogrammar utilized MPC control with a similar, though slightly different, reward function for evaluating its designs, whereas our method employs a model-free RL approach. Finally, `RoboMorph` is evaluated in a different simulator (MJX) than Robogrammar (PyBullet). While the underlying physics solvers of both simulators should be largely similar, this difference is worth acknowledging.

5 Related Work

LLMs for design. Limited work has been conducted in the area of LLM-based robot design, which is arguably a bottleneck in the field Zeng et al. (2023). The work in Stella et al. (2023) introduces a framework that utilizes ChatGPT to guide the robot design process at both the conceptual and technical levels with questions prompted by a human-in-the-loop. An extensive study Makatura et al. (2023) presents the use of LLMs for design and manufacturing, leveraging GPT-4's ability to generate high-level structures through discrete compositions. Unlike previous studies, our approach integrates a fitness evaluation module within the framework, enabling optimization of the design for specific objectives. The most similar work to ours is Text2Robot Ringel et al. (2024), which presents a framework for converting user text specifications into quadrupedal robot designs. However, our approach addresses a key limitation of Ringel et al. (2024) by enabling the design of morphologies with a varying number of limbs and joints.

Legged locomotion. Designing control algorithms for legged locomotion has been a long-standing challenge in robotics due to the high-dimensional state space and the system's nonlinear dynamics. Model Predictive Control (MPC) Di Carlo et al. (2018); Kim et al. (2019), as used in Robogrammar, along with classical control methods, often requires precise system dynamics models and is challenging to tune for complex systems. Consequently, we adopt a model-free approach using reinforcement learning Hwangbo et al. (2019); Haarnoja et al. (2018a); Tsounis et al. (2020); Lee et al. (2020), specifically leveraging the Soft Actor-Critic (SAC) algorithm Haarnoja et al. (2018b) to learn policies for evaluating robot designs.

6 Limitations and Future Work

More tailored input prompts. A valuable direction for future work is to develop a curated set of input prompts specifically tailored to robot design. This would require further advancements in prompt engineering to refine the LLM's output responses. One promising approach could involve incorporating mutation operators commonly found in EAs into the natural language space, as demonstrated in Fernando et al. (2023). This may be advantageous since the natural language space operates at a lower fidelity than the corresponding design space, potentially making search and optimization more efficient. Additionally, automating prompt discovery could help mitigate occasional hallucinations produced by the LLM for the given task.

Diverse mix of environments. A natural next step is to ensure broad coverage of various environmental features, such as terrain shape and texture. Extending this work to design a more general-purpose robot capable of traversing multiple terrains simultaneously would be an interesting direction. An open question remains whether this could be achieved by initializing the system with diverse few-shot examples specifically tailored to different environments. Alternatively, incorporating textual descriptions of the environment into the prompt may encourage the LLM to generate morphologies that dynamically adapt to varying terrain conditions.

Joint generation of a robot and its policy. Future research could explore whether LLMs are capable of co-designing both a robot's morphology and its control strategy. Recent work, such as Ma et al. (2024a), has demonstrated the possibility of using LLMs to generate reward functions without task-specific prompting or predefined reward templates.

7 Conclusion

We present `RoboMorph`, a framework that integrates LLMs, evolutionary algorithms, RL-based control, and robot grammars to facilitate the design of modular robots. Our approach streamlines and simplifies the workflow of conventional robot design methods while enabling more efficient design schemes. Experimental results demonstrate significant improvements in robot morphology over successive evolutions. Future research will further explore the intersection of LLMs' generative capabilities and low-cost additive manufacturing techniques to develop robots suitable for real-world deployment.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023a.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023b.

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

Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 1–9. IEEE, 2018.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL `http://github.com/google/brax`.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018b.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

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.

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, 2024a.

Yecheng Jason Ma, William Liang, Hungju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. 2024b.

Liane Makatura, Michael Foshey, Bohan Wang, Felix HähnLein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, et al. How can large language models help humans in design and manufacturing? *arXiv preprint arXiv:2307.14377*, 2023.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

Ryan P Ringel, Zachary S Charlick, Jiaxun Liu, Boxi Xia, and Boyuan Chen. Text2robot: Evolutionary robot design from text descriptions. *arXiv preprint arXiv:2406.19963*, 2024.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

Murray Shanahan. Talking about large language models. *Communications of the ACM*, 67(2):68–79, 2024.

Francesco Stella, Cosimo Della Santina, and Josie Hughes. How can llms transform the robotic design process? *Nature Machine Intelligence*, pp. 1–4, 2023.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy, 2024.

Dirk Thierens. Scalability problems of simple genetic algorithms. *Evolutionary computation*, 7(4): 331–352, 1999.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706, 2020.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Fanlong Zeng, Wensheng Gan, Yongheng Wang, Ning Liu, and Philip S Yu. Large language models for robotics: A survey. *arXiv preprint arXiv:2311.07226*, 2023.

Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

A Initialization of Few-Shot Examples

The pseudocode for initializing the few-shot examples is presented in Algorithm 2 below.

Algorithm 2 Pseudocode for initializing few-shot examples

```
Requires:
prompt_system                                          robot grammar rules
max_steps                                          maximum number of steps
$K$                                               number of few-shot examples

function init_few_shots
fewshots ← $\emptyset$
repeat
    Initialize design ← $\emptyset$
    Initialize $steps \leftarrow 0$
    while $steps <$ max_steps do
        Identify structural rules $R$ from prompt_system
        if $R = \emptyset$ then break
        Sample $r \sim R$ and apply rule $r$ to list design
        $steps \leftarrow steps + 1$
    end while
    for all $r_i \in$ design do
        Identify components $C$ for $r_i$ from prompt_system
        Sample $c \sim C$ and swap $r_i$ with $c$ in design
    end for
    fewshots.append(str(design))
until |fewshots| $= K$
return fewshots
end function
```

B Prompts

All RoboMorph prompts are highlighted below.

Prompt 1: System Prompt

```
You are a helpful assistant specialized in designing robots.
My ultimate goal is to discover as many diverse designs as
possible, accomplish as many diverse tasks as possible and
become the best robot designer in the world.


You will represent robot designs as graphs composed of
interconnected nodes, where each node corresponds to a
specific component or function of the robot.  You must
construct these graphs by applying structural rules
sequentially and then replacing the nodes with specific
component rules to generate a final design.  The process
ensures the robot is both structurally valid and functionally
robust.


The following symbols represent the nodes in the robot graph:
S: Start symbol
H: Head part
Y: Body joint
B: Body part
T: Tail part
```

```
U: Body link
C: Connector
M: Mount part
E: Limb end
J: Limb joint
L: Limb link

STRUCTURAL RULES:
Start node
r0) S

Body structure
r1) Replace S with H-B-T
r2) Replace T with Y-B-T

Adding appendages to the body
r3) Replace B with U-(C-M-E)
r4) Replace B with U

Appendages
r5) Replace E with J-L-E
r6) Replace T with C-M-E
r7) Replace H with E-M-C

COMPONENT RULES:
U: body={15cm}
L: limb={10-15cm}
Y: rigid, roll or twist
J: rigid, roll, knee={0-60deg} or elbow={0-180deg}
C: connector
M: mount
E: wheel or null
H: null
T: null

Note:  Component rules are applied only after the structural
graph is fully constructed.  Each node must be replaced with
its corresponding component, selecting exactly one parameter
from each {} set.

Apply the structural rules sequentially.  For each step,
specify the rule applied (e.g., r1, r2) and display the
updated graph at each step, clearly indicating node
replacements.  Once the structural graph is complete (all end
nodes are H, T, or E), replace each node with its respective
component as per the component rules.

Use the following format for responses:

STRUCTURAL RULES:
Step 1:  r0) S
Step 2:  r1) H-B-T
Step 3:  ...
```

```
...

COMPONENT RULES:
Final graph representation with components.

REASONING: Provide an explanation of your design choices,
including why specific rules were applied.

Here are some important considerations:
Completeness:  Ensure all end nodes are resolved into H, T, or
E before applying component rules.
Consistency:  Adhere to the defined structural and component
rules.
Diversity:  Prioritize generating a wide range of designs by
exploring different rule combinations.
```

Prompt 2: User Prompt

```
Your task is to design a single robot.  Use the examples
provided to guide your reasoning and explain your design step
by step.
```

Prompt 3: Few Shot Examples

```
### EXAMPLE {i} ###
STRUCTURAL RULES:
Step 1:  r0) S
Step 2:  r1) H-B-T
Step 3:  ...

COMPONENT RULES:
[null]-[body=15cm] ...

REASONING: ...
```

C Hyperparameters and Compute

For reinforcement learning, we use the SAC algorithm implementation available in Brax, with hyperparameters listed in Table 1.

Table 1: Reinforcement learning hyperparameters

Hyperparameter	**Value**
`num_envs`	4096
`num_timesteps`	1,000,000
`max_replay_size`	4,000,000
`min_replay_size`	5,000
`episode_length`	1,000
`discounting`	0.99
`num_envs`	1024
`batch_size`	256
`unroll_length`	62
`learning_rate`	1e-3
`hidden layers (for both critic and actor)`	[256,256]
`grad_updates_per_step`	64
`num_minibatches`	16
`entropy_cost`	0.0

D Robogrammar Best Designs

Figure 7 presents our reconstruction of the best designs for the ridged, flat, and frozen terrains using the Robogrammar method Zhao et al. (2020).

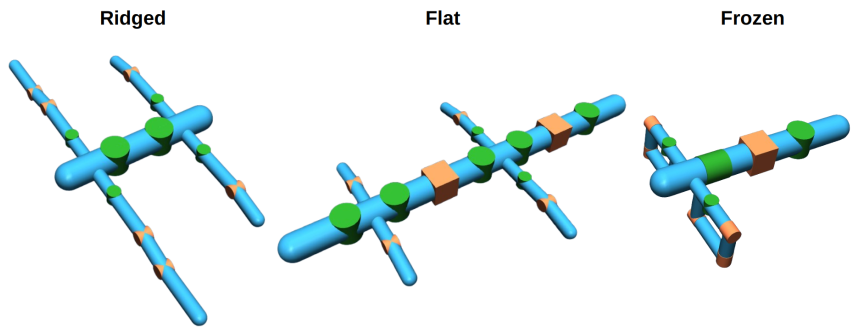


Figure 7: Best-performing designs generated by Robogrammar for the ridged, flat, and frozen terrains.