Interactive LLM-assisted Curriculum Learning for Multi-Task Evolutionary Policy Search

Berfin Sakallioglu
MIGe - University of Trieste
Trieste, Italy
berfin.sakallioglu@phd.units.it

Giorgia Nadizar
University Toulouse Capitole, IRIT - CNRS UMR5505
Toulouse, France
giorgia.nadizar@irit.fr

Eric Medvet
DIA - University of Trieste
Trieste, Italy
emedvet@units.it

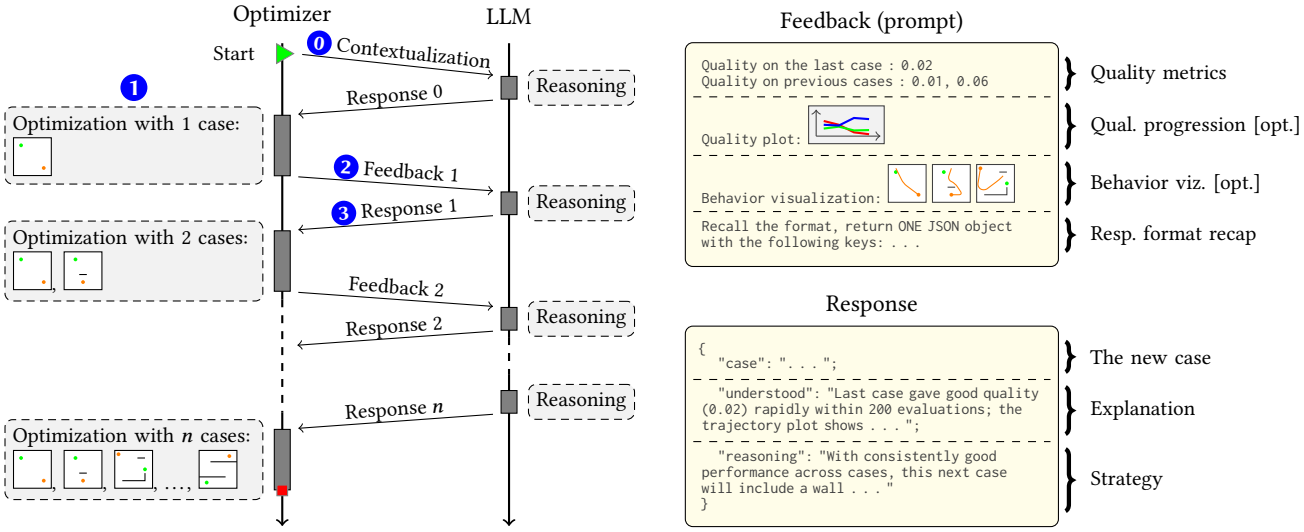

Figure 1: An overview of our fully unsupervised pipeline for performing curriculum learning (CL) policy search using a curriculum generated by an LLM based on the feedback on the optimization process. At the beginning (▶) the optimizer 0 sends a contextualization prompt to the LLM describing the general context, the subsequent interactions, and the format of feedback and response messages, and asking for an initial training case. Then, the optimizer repeats the following steps, starting with a set comprising the initial LLM-provided training case: 1 it performs an optimization run, 2 it sends a summary of this run to the LLM, asking for a new case, 3 it gets a response with a new case and adds it to the training set. After n iterations of 1 2 3, the pipeline stops (■) and the optimizer returns the best policy found in the process. On the right, we sketch the structure of the feedback prompt and the LLM response, highlighting the salient components.

Abstract

Multi-task policy search is a challenging problem because policies are required to generalize beyond training cases. Curriculum learning has proven to be effective in this setting, as it introduces complexity progressively. However, designing effective curricula is labor-intensive and requires extensive domain expertise. LLM-based curriculum generation has only recently emerged as a potential solution, but was limited to operate in static, offline modes without leveraging real-time feedback from the optimizer. Here we propose an interactive LLM-assisted framework for online curriculum generation, where the LLM adaptively designs training cases based on real-time feedback from the evolutionary optimization process. We investigate how different feedback modalities, ranging from numeric metrics alone to combinations with plots and behavior visualizations, influence the LLM ability to generate meaningful curricula. Through a 2D robot navigation case study, tackled with genetic programming as optimizer, we evaluate our approach against static LLM-generated curricula and expert-designed baselines. We show that interactive curriculum generation outperforms static approaches, with multimodal feedback incorporating both progression plots and behavior visualizations yielding performance competitive with expert-designed curricula. This work contributes to understanding how LLMs can serve as interactive curriculum designers for embodied AI systems, with potential extensions to broader evolutionary robotics applications.

GECCO '26, San Jose, Costa Rica

ACM ISBN 979-8-4007-2487-9/2026/07
https://doi.org/10.1145/3795095.3805152

CCS Concepts

• **Computing methodologies** → Genetic programming; *Active learning settings*; • **Information systems** → **Language models**.

Keywords

Large language models, Quality diversity, Genetic programming

ACM Reference Format:

Berfin Sakallioglu, Giorgia Nadizar, and Eric Medvet. 2026. Interactive LLM-assisted Curriculum Learning for Multi-Task Evolutionary Policy Search. In *Genetic and Evolutionary Computation Conference (GECCO '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3795095.3805152

1 Introduction and related works

Multi-task (MT) learning trains a single model to solve several related tasks jointly instead of in isolation, encouraging shared representations that improve generalization to new but similar tasks [7, 48]. In robotics, for example, a visuomotor policy might be trained across families of manipulation tasks such as pushing objects with different shapes, opening doors and drawers with varying kinematics, or placing items into containers at different poses; these tasks share perception and embodiment but differ in contact dynamics and success criteria [10, 45]. Learning them in an MT setting is particularly important for real-world deployment, where agents might be called to rapidly adapt to novel tasks that are close variants of those seen during training, rather than repeatedly relearning from scratch [16].

Evolutionary policy search provides a general framework for optimizing MT policies without relying on gradients. By iteratively selecting, mutating, and recombining candidate solutions, evolutionary policy search can optimize arbitrary policy representations, including interpretable [8, 33] or structured policies discovered via genetic programming (GP) [40]. This flexibility makes evolutionary optimization particularly interesting for MT settings, where diverse [31, 32] or structured [9] solutions may be needed. For example, GP has been applied in MT visual learning domains [18]—such as evolving controllers that perform well across multiple Atari games or diverse image processing tasks [5, 6]—demonstrating that symbolic policies can capture shared structure while remaining potentially human-interpretable.

The iterative nature of evolutionary policy search naturally interacts with curriculum learning (CL) [20, 21, 24]: by controlling how and when tasks (or data from each task) are presented, curricula can guide populations toward solutions that capture shared structure more efficiently [4]. In MT settings, curricula often stage tasks so that simpler or information-rich tasks bootstrap learning of features or structures that later support harder problems [35].

However, designing effective curricula poses a fundamental challenge. Crafting representative, difficulty-calibrated training instances or auxiliary tasks typically requires substantial domain expertise [4, 37]. In robotics, this often entails manually creating synthetic trajectories or intermediate challenges to target specific capability gaps, frequently without access to real-world data [12]. Moreover, even when suitable tasks are available, scheduling them effectively is nontrivial [27].

Adaptive curricula offer a promising alternative by adjusting task sampling online based on learning progress, validation performance, or model uncertainty, focusing training on tasks that are neither too easy nor too difficult [14, 41]. Such approaches can improve transfer, aligning training pressure with the agent's evolving capabilities to enhance generalization [19, 27, 42, 44]. Yet jointly designing task content and scheduling—especially when done by humans—adds significant complexity, requiring careful supervision and continual intervention during training [37].

In this context, large language models (LLMs) provide a promising path toward fully automated curriculum design. They can generate diverse tasks, environments, reward functions, or curricula, either statically [26, 36] or adaptively [11, 22, 23]. LLMs have been used to propose initial environments for training [23], dynamically adjust difficulty based on agent performance [22], and even produce structured teaching curricula in educational settings [47]. Interactive LLM-based CL approaches, such as EnvGen [46], combine task generation with feedback from the agent, allowing the curriculum to progress alongside the learner.

Building on these advances, we propose interactive LLM-assisted curriculum learning for MT evolutionary policy search. Our approach uses an LLM to generate candidate tasks or task variants, receive feedback from evolutionary optimization signals, and iteratively refine the curriculum to match the population's evolving capabilities. Unlike prior methods [46], we never train agents on hand-designed test cases. In addition, our method supports potentially interpretable controller representations and explores different feedback modalities, including multimodal signals, which can further improve learning [17].

We test our approach on a 2D navigation task using a symbolic equation-based controller and find that interactive LLM-generated curricula outperform static LLM-generated curricula, with richer feedback further boosting performance. Overall, our method achieves results comparable to expert-designed curricula with zero human effort, demonstrating that it can efficiently guide MT learning, even for interpretable representations. Moreover, this also opens the door to larger-scale applications, as LLMs can inherently generate a wide variety of environments [11] and can even be fine-tuned to produce specialized ones [1, 38].

2 Interactive LLM-assisted curriculum learning

We propose a fully unsupervised pipeline for performing CL policy search using a curriculum generated by an LLM based on the feedback concerning the optimization process. The pipeline, sketched in Figure 1, involves two actors, the *optimizer* and the *LLM*, interacting iteratively through messages for a given number of times.

We assume that the problem being tackled can be instantiated in *cases*, *i.e.*, problem instances on which a candidate policy can be evaluated. Moreover, we assume that the optimizer can perform the optimization on a bag of one or more cases, called training cases.

The optimizer-LLM interaction starts with a ⓪ *contextualization message* sent by the optimizer to the LLM to provide basic information and context about the overall interaction and to ask for an initial training case c_0. This message serves for instructing the LLM about the subsequent interactions, to describe the format of feedback messages, and to prescribe the format of responses, namely on how to encode new cases.

After this first exchange, the interaction works as follows, starting from a bag $C = \{c_0\}$ containing the single initial training case provided by the LLM. For a given number n_{stage} of times, ① the optimizer runs an optimization run on C, ② the optimizer sends

a *feedback message* to the LLM containing the feedback about the optimization run, ❸ the LLM responds with a message containing a new case c, then the bag of training cases is augmented with c, $C \leftarrow C \oplus \{c\}$. We call *stage* an iteration of the three steps above. We call *curriculum* the sequence of training bags obtained by the optimizer at the end of the process.

The precise format of messages (*i.e.*, the corresponding *prompts*) sent by the optimizer is problem-specific. We describe them in detail later for the case study which we consider for assessing our proposal. Figure 1 (on the right) sketches the structure of these messages, highlighting the main components; Appendix A shows them in detail.

We remark that, at each stage, the bag C includes the cases of all the previous stages: we opt for accumulating training cases, rather than training at each stage on a single case, for avoiding catastrophic forgetting, *i.e.*, having the optimizer producing a policy that solves the current case but "forgets" how to solve the previous ones, eventually hindering generalization [13]. As an aside, this relieves the user from the burden of choosing an optimizer and a policy representation which are robust to catastrophic forgetting.

2.1 Feedback modality

We investigate three feedback modalities that progressively increase information richness:

Numeric feedback (N) contains only quantitative performance metrics, namely, the quality scores of the policy found by the optimizer assessed on each case in C.

Progression-augmented feedback (N+P) extends N by adding a visualization of optimization convergence. This plot shows the quality of the best policy across optimization iterations (x-axis: iteration number; y-axis: quality metric), one line for each case in C, revealing how quickly and effectively the optimization process improved.

Behavioral-augmented feedback (N+P+B) further extends N+P by including a visualization of the policy behavior on the training cases. This provides the LLM with direct insight into the policy behavioral characteristics and how it interacts with the problem constraints. In the case study considered here (see Section 3), behavior is visualized as a spatial trajectory showing the path taken by the agent overlaid on the environment layout.

2.2 Requirements and limitations

Our pipeline is general and agnostic with respect to the inner working of the optimizer, the nature of the policy, and the considered problem. Nevertheless, there are some requirements concerning the optimizer and the problem.

The optimizer must: (a) accept multiple training cases that accumulate over stages, (b) produce a numeric quality measure quantifying the policy performance on each case in C, and (c) if N+P or N+P+B feedback is employed, track and report the quality progression across optimization iterations for each case. Moreover, it is convenient, but not strictly required, that the optimizer can start an optimization from the results of a previous optimization run: this makes the entire process more efficient and effective.

The problem must: (a) allow for a case representation which can be encoded in a format the LLM can generate and (b) if N+P+B feedback is employed, allow for a visual representation of the agent behavior on a case, given a policy. The first problem requirement might appear hard to meet, as the syntax describing a case might be custom and not known to the LLM. However, current LLMs are becoming increasingly better at complying with provided guidelines [39, 43]. And there are also some LLMs which have been customized for specific domains, e.g., MarioGPT [38] used for generating levels of the Super Mario Bros game in [21]. Moreover, in many cases one can develop a procedure that automatically fixes small issues in LLM-generated cases. In the problem considered as case study in this work, the LLM was in general able to generate syntactically correct cases.

3 Case study: 2D navigation

We consider as use case the problem of learning a policy that permits a simulated robotic agent to navigate *any* 2D arena avoiding obstacles and reaching a target which can be perceived by the agent through sensors.

3.1 Model, policy, and optimization

3.1.1 Agent and environment. The agent is a simulated differential-drive robot with two wheels, five proximity sensors, and two sensors for sensing the distance to and the relative direction of the target. The environment is a 2D arena defined by a robot starting position, a target position, and some obstacles in the form of walls placed inside the arena. The size of the arena is of $1\,\text{m} \times 1\,\text{m}$; the robot has a circular shape with a radius of $0.02\,\text{m}$.

We simulate the movement of the robot in the arena in discrete time with a time step of $0.1\,\text{s}$. At each time step, the robot perceives the distance to the closest obstacle (arena internal and external walls) along five directions equally spread on the front side of the robot (*i.e.*, $-\frac{\pi}{2}$, $-\frac{\pi}{4}$, 0, $\frac{\pi}{4}$, and $\frac{\pi}{2}$), the distance to the target, and the relative direction to the target. The proximity sensors and the distance-to-target sensor have a range of $0.5\,\text{m}$: if there are no objects in the range, the sensors sense $0.5\,\text{m}$; the domain of these sensors readings is hence $[0, 0.5]$. The angle-to-target sensor domain is $[-\pi, \pi]$.

Upon the processing of the sensory input, the robot sets independently the rotational speed of the two wheels in $[-\omega_{\max}, \omega_{\max}]$. We set $\omega_{\max}$ such that the maximum linear speed of the robot is $0.01\,\text{m}\,\text{s}^{-1}$. When the robot collides with an obstacle, the linear speed goes to zero but we allow it to rotate.

3.1.2 Policy representation. Given the sensors and actuators of the robot, its controller is in general a dynamical system with $\mathbb{R}^7$ as observation space and $\mathbb{R}^2$ as action space. We implement the controller as an outer fixed part wrapping an inner variable part, which is the one we actually optimize.

The outer part pre-processes the observation for the wrapped inner part and post-processes the latter output before using it as actuation values for the wheels. In the pre-processing, it takes the original observation $\boldsymbol{o}^{(k)} \in \mathbb{R}^7$ and (i) normalizes each element in the proper domain (*i.e.*, $[0, 0.5]$ or $[-\pi, \pi]$) and (ii) augments it with trend and average value during the last $0.5\,\text{s}$: *i.e.*, for each $o_i^{(k)}$, the augmented

observation $\boldsymbol{o}'^{(k)} \in [-1,1]^{21}$ contains normalized $o_i^{(k)}$, $o_i^{(k)} - o_i^{(k-4)}$, and $\frac{1}{5}\sum_{j=0}^{j=4} o_i^{(k-j)}$. In the post-processing, the outer part takes the output $a' \in \mathbb{R}$ of the inner part and translates it to $\boldsymbol{a}^{(k)} \in [-\omega_{\max}, \omega_{\max}]$, with $a_1^{(k)} = a_{\text{left}}^{(k)} = \omega_{\max}\left(1 + 2\min\left(0, \tanh a'^{(k)}\right)\right)$ and $a_2^{(k)} = a_{\text{right}}^{(k)} = \omega_{\max}\left(1 - 2\max\left(0, \tanh a'^{(k)}\right)\right)$. This way, if $a'^{(k)} \to -\infty$ then $\boldsymbol{a}^{(k)} = (-\omega_{\max}, \omega_{\max})$ and the robot rotates to the left, if $a'^{(k)} \to +\infty$ then $\boldsymbol{a}^{(k)} = (\omega_{\max}, +\omega_{\max})$ and it rotates to the right, if $a'^{(k)} = 0$ then $\boldsymbol{a}^{(k)} = (\omega_{\max}, \omega_{\max})$ and it goes straight at the maximum speed. In the intermediate cases, the robot follows a curvy path, with the radius of the curve depending on $|a'^{(k)}|$. Note that the outer part of the controller is stateful in the pre-processing, as it maintains a memory of the last 5 observations, and stateless in the post-processing.

As inner part of the controller we use a function in $\mathbb{R}^{21} \to \mathbb{R}$ which we encode through a syntax regression tree, suitable to be optimized with GP. Namely, we use the following 11 labels for the non-terminal nodes: $\bullet + \bullet$, $\bullet - \bullet$, $\bullet \times \bullet$, $\bullet \div^* \bullet$, $\log^* \bullet$, $\max(\bullet, \bullet)$, $\min(\bullet, \bullet)$, $\tanh \bullet$, $\bullet > \bullet$, $\bullet < \bullet$, $\bullet ? \bullet : \bullet$, where bullets represent the children of the corresponding node and hence dictate its arity. $\div^*$ and $\log^*$ represent the protected versions of the corresponding operators. The comparison operators $>$ and $<$ return the sign of the difference of the two arguments; $\bullet ? \bullet : \bullet$ corresponds to the ternary operator where the first argument is compared against 0. For the terminal nodes we use as labels the observation variables $o_1'^{(k)}, \ldots, o_{21}'^{(k)}$ and ephemeral constants in $[-5, 5]$.

Summarizing, we drive the robotic agent with a GP tree encoding a function from $\mathbb{R}^{21}$ to $\mathbb{R}$ wrapped within a dynamical system which augments sensor readings with "historical" information and simplifies the action space. We call policy the inner tree.

3.1.3 Tasks. The overall goal is to find a policy allowing a robot to navigate any arena, *i.e.*, able to solve the general problem of reaching a target while avoiding collisions. We attempt to achieve this goal by making candidate policies experience different arenas, each one constituting a *training case*. We call episode a simulation of a robot with a policy within an arena. In our settings, an episode is deterministic.

Given an arena with a starting position for the robot and a position for the target, we run an episode for 60 s (simulated time, *i.e.*, for 600 time steps). At the beginning of the episode, we set the direction of the robot to the right, regardless of the position of the target. At the end of episode, we record the full trajectory of the robot. Out of this, we compute several metrics, including the final (*i.e.*, at $k = 600$) and average distance of the robot to the target.

We remark that this 2D navigation problem has been widely used in previous work as a basic, yet not trivial, benchmark for policy search, often for assessing quality diversity (QD) search algorithms [3, 15, 25, 28]. However, in most of the cases, policies were assessed only on the arena they were trained on. Instead, here we aim at obtaining a policy able to navigate, in general, any arena, which is a much harder problem. *I.e.*, we cast this navigation problem as a *multi-task (MT)* problem where a single policy is required to solve many (similar) tasks, each one defined by an arena.

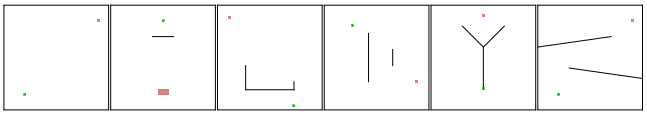

Figure 2: The six test arenas used for assessing policies. The red region shows the robot starting position (which is placed at that center of the region), the green region shows the target position.

To assess the effectiveness of a policy to solve the MT navigation problem, we consider the set C_{test} of six test arenas of Figure 2. We remark that these six test arenas are never known to the optimizer, nor to the LLM.

3.1.4 Evolutionary optimizer. Based on the recent literature involving the 2D navigation problem, we use MAP-elites (ME) as optimizer. "Classic" ME [30] evolves a population of candidate solutions using only mutation as variation operator. Solutions compete to occupy a cell in a discrete structure called archive: ME maps solutions to cells using two descriptors which provide numerical coordinates which are then discretized. Here we use a grid-based archive, where discretization corresponds to matching continuous values of each descriptor against equally sized bins in the corresponding domain.

Given a search space S (here, the set of GP trees used as policies), a fitness function $f : S \to \mathbb{R}$, and two descriptors $d_1 : S \to D_1 \subset \mathbb{R}$ and $d_2 : S \to D_2 \subset \mathbb{R}$, our ME works as follows—we assume that f has to be minimized.

First, we initialize a population of n_{pop} candidate solutions and insert each solution in the archive. To this end, for each solution s, we compute its coordinates $\boldsymbol{d} = \left(d_1(s)|_{D_1,n_{\text{bins}}}, d_2(s)|_{D_2,n_{\text{bins}}}\right)$ in the archive, where $x|_{X,n} = \left\lfloor n \frac{x - \min_{x' \in X} x'}{\max_{x' \in X} x' - \min_{x' \in X} x'} \right\rfloor \in \{0, \ldots, n\}$ denotes the discretization of x using n equally sized bins in $X \subset \mathbb{R}$. If the cell is empty, we put s in the archive. If the cell contains a solution s', we replace it with s only if $f(s) \leq f(s')$. After the initialization, we repeat the following steps until some termination criterion is met. (i) We generate an offspring of n_{batch} new solutions by extracting, randomly with repetitions, n_{batch} solutions from the archive and mutating each one of them. (ii) We possibly insert each new individual in the archive, as for the initial population. At the end of the evolution, the archive contains a set of solutions, *i.e.*, policies, which are diverse based on the descriptors and are good based on f.

For initializing the population at the first stage, we use the ramped-half-and-half procedure, commonly used in GP, with tree depth enforced to be in $[4, 10]$. For initializing the population at subsequent stages, we take the best $\frac{1}{4} n_{\text{pop}}$ solutions of the last iteration population of the previous stage, one mutated solution obtained from each of them, and generate the remaining $\frac{1}{2} n_{\text{pop}}$ solutions randomly with the ramped-half-and-half procedure: this way, we seed the initial population of the i-th stage with the outcome of the optimization of the $i - 1$-th stage.

As mutation, we use with equal probability the GP subtree mutation or a Gaussian perturbation of the terminal nodes labeled with numerical constants, with a $\sigma = 0.25$: when mutating trees, we enforce a limit on the tree depth ≤ 10.

Fitness function and descriptors. Given a bag C of training cases (here, arenas) and a candidate solution s (here, a tree), we compute the fitness as the weighted average of the final and average distance of the robot to the target, averaged across arenas, *i.e.*, $f(s;C) = \frac{1}{|C|}\sum_{c\in C} f_{\text{single}}(s;c)$, where:

$$f_{\text{single}}(s;c) = 0.9\left\|\boldsymbol{x}_{\text{robot}}^{(600)} - \boldsymbol{x}_{\text{target}}\right\| + 0.1\frac{1}{600}\sum_{k=1}^{k=600}\left\|\boldsymbol{x}_{\text{robot}}^{(k)} - \boldsymbol{x}_{\text{target}}\right\|$$

is the weighted average on a single arena c, with $\boldsymbol{x}_{\text{robot}}^{(k)}$ being the position of the robot equipped with s at step k of an episode in the arena c and $\boldsymbol{x}_{\text{target}}$ being the target position in c. We choose to use an average of the final and average distance to the target, and not just the final distance, for driving the optimization, as this leads to "smarter" behaviors, awarding policies resulting in shorter trajectories to the target. However, we assign a relatively low weight (0.1) to the average distance as we verified that larger values make the search harder, as certain arenas may require to increase the length of the path (and hence worsen the average distance) to overcome a local optimum.

As ME descriptors, we considered the gap along the x and y axes of the robot to the target at the end of the episode, averaged across arenas, *i.e.*, $d_1(s;C) = \frac{1}{|C|}\sum_{c\in C}\left(x_{1,\text{robot}}^{(600)} - x_{1,\text{target}}\right)$ and $d_2(s;C) = \frac{1}{|C|}\sum_{c\in C}\left(x_{2,\text{robot}}^{(600)} - x_{2,\text{target}}\right)$. These descriptors correspond to the adaptation of the classic ME descriptors used for the 2D navigation problem on a single arena to the case where a policy is assessed on many arenas. In the classic case, the final coordinates of the robot are used as descriptors: this would not make sense here as different arenas have in general different target position, obstacles, and starting position. We hence consider the relative position to the target and average it across arenas. We set the descriptor domains to $D_1 = D_2 = [-0.5, 0.5]$.

Parameters and remark. Based on experience, previous work, and exploratory experiments, we set the parameters of the optimizer as follows: $n_{\text{pop}} = n_{\text{batch}} = 100$, $n_{\text{bin}} = 10$, and, as termination criterion, the number of fitness evaluations being performed (which we set to 10 000 or more depending on the experiment). We recall that a single fitness evaluation corresponds to performing $|C|$ episodes.

We verified experimentally that this combination of policy representation and optimizer is effective enough to tackle the MT 2D navigation problem. In particular, we compared it against combinations of genetic algorithm (GA) as optimizer, artificial neural networks (ANNs) as policies, and different ME descriptors based on both the behavior or on policy features: we found that ME with the descriptors based on the relative final position and GP trees for representing the policy is the best combination. We remark, however, that the main contribution of this work is the optimizer-LLM interaction scheme, which is by design agnostic to these lower level details.

3.2 LLM interaction in the 2D navigation case

The overall interaction scheme described in Section 2 is general, and many components of the prompt remain domain-agnostic. However, certain elements are problem-dependent and require customization for the 2D navigation task. Here we describe these problem-specific adaptations.

As shown in Figure 1, we begin with a contextualization prompt that differs from subsequent feedback prompts. This initial prompt establishes the 2D navigation domain, defining the differential drive robot task where the agent must navigate from a start position to a target while avoiding walls. The prompt specifies the role of the LLM as a stage designer, defines the quality metric (distance to target), describes the feedback format it will receive, enforces validity constraints, and provides an example arena. We give the complete contextualization prompt in Appendix A.

We communicate arena configurations through character string encodings. We assume that an arena is described by tiles and each tile is encoded as a character: `s` represents the start position, `t` the target, `w` denotes walls, and `e` indicates empty tiles. The LLM generates 15×15 character grids with rows separated by the `|` character. A substantial portion of the prompt enforces structural constraints, including exactly one start and one target position, guaranteed reachability via a continuous path, consistent row lengths of 15 characters, and internal validation of connectedness. The complete format specification is visible in the prompt shown in Appendix A..

Subsequent feedback prompts vary by modality. For N, we provide quality scores including the best policy distance to target on the current case, historical performance on previous cases, and average quality across all accumulated cases. N+P extends this with a convergence plot (PNG format) showing distance to target (y-axis) over fitness evaluations (x-axis), with separate lines for each accumulated case to reveal optimization dynamics. A legend identifies each case line in the plot. N+P+B further adds trajectory visualizations—2D plots (SVG format) similar to Figure 3 displaying walls, start position, target, and the spatial path taken by the best-performing policy at the end of training for each case. A feedback example showing the convergence plot and the trajectory visualization is shown in Section 4.4; a complete interaction sequence is presented in Appendix A.

4 Experimental Evaluation

We evaluate the proposed pipeline through a series of experiments comparing different feedback modalities and baseline approaches. Our experimental design addresses the following research questions:

- RQ1 Does the pipeline enable learning, with policies improving across curriculum stages and generalizing to unseen cases? How do LLM-generated curricula compare to expert-designed, random, and static baselines? Which feedback modality is the most effective among the three investigated conditions?
- RQ2 Are LLM-generated curricula actual curricula? Do their cases work equally well if administered in a single batch instead of progressively?
- RQ3 Does the LLM really "understand" the feedback? Does it exploit it when creating new cases?

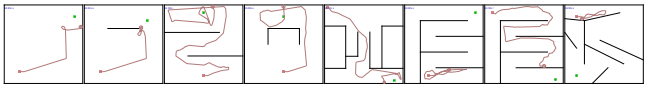

Figure 3: The expert-designed curriculum with the trajectories obtained with a policy optimized on this curriculum.

4.1 Experimental procedure and baselines

We compare three versions of our pipeline corresponding to the feedback modalities described in Section 2.1 against three baselines without feedback.

Expert Represents an expert-designed static curriculum of eight cases (see Figure 3) created by a domain expert (one of the authors of this paper), serving as our gold standard. We generated this curriculum based on our (quite long) experience on the navigation problem and evolutionary approaches for solving it; we also knew the test arenas.

Static Consists of an LLM-generated curriculum which we obtain by asking the LLM to build eight cases in a single batch without iterative feedback, using a modified prompt.

Random Comprises randomly generated cases with randomized wall segments of varying lengths, numbers, and orientations, as well as random start and target positions.

All curricula have the same number of cases (eight). We administer all of them progressively, in eight stages: first stage with the first arena, second with the first and the second arenas, and so on.

We repeat the execution of each method 10 times to assess statistical significance and consistency. For the variants (both with and without feedback) involving the LLM, we execute the entire process 10 times, resetting the LLM session at each repetition. For the random baseline, we generate 10 random curricula. For the expert baseline, we execute 10 sequences of optimization runs with different random seeds on the same expert-designed curriculum.

We use JGEA [29] for performing the optimization with ME and for simulating the navigation episodes. We use Claude Sonnet 4.5 [2] with extended thinking capabilities as the LLM—we leave to future work an investigation on the impact of using other LLMs. We forwarded the messages between the optimizer (JGEA) and the LLM (Claude) manually, *i.e.*, by copy-pasting JGEA output within templated prompt and inserting LLM responses within JGEA experiment description file. This way we are able to monitor in real-time the correct execution of the interaction.

The execution of one interaction with a variant with feedback lasts on average 20 min: ≈ 10 % is taken by the LLM to respond (with no significant variance across feedback modalities), the remaining by the optimizer—we run it on a machine equipped with an Intel® Core™ i5 14500T vPro® CPU (14 cores at 4.8 GHz, with 20 threads) and 16 GB of RAM.

The code for reproducing our experiments is available at https://github.com/berfins/CL-LLM.

4.2 General effectiveness (RQ1)

Figure 4 presents the learning dynamics across all methods over eight curriculum stages, showing both *training performance* (top row) and test *generalization* (bottom row). Namely, the figure shows

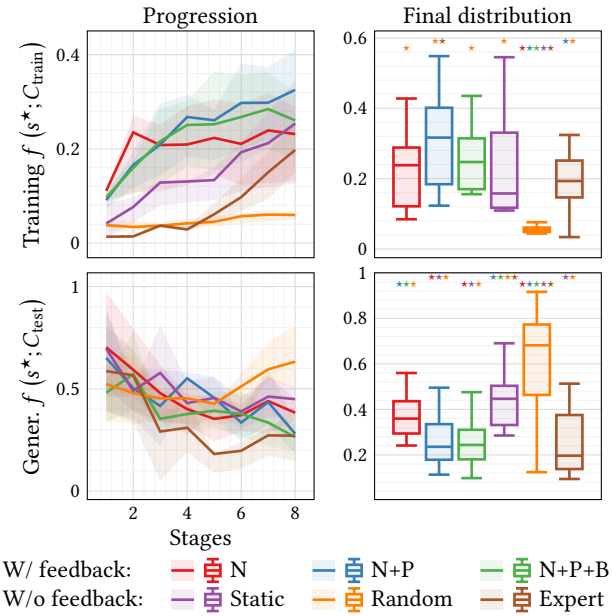

Figure 4: Progression (right plots) and final distribution (left plots) of the performance of the best policy $s^\star$ in the population measured on the train arenas (top plots) and on the test arenas (bottom plots), for the three interactive modalities and the three baselines. In the progression plots, the line corresponds to the median value across the 10 repetitions, the shaded area to the interquartile range. In the distribution plots, stars above the boxes show significant differences: e.g., an orange star over the red box indicates that N is significantly different than Random.

the value of f (weighted average of final and average distance to target) computed for the best policy $s^\star$ in the population respectively on the train (C_{train}) or test (C_{test}) arenas—the lower, the better. The left column shows progression across stages as line plots (median and interquartile range over ten repetitions), while the right column shows distribution of the final training and test performance.

The line plots reveal two patterns. For training performance, performance $f(s^\star; C_{\text{train}})$ generally increases across stages for all methods except Random —, indicating that curricula progressively introduce harder cases as intended. For test performance, performance $f(s^\star; C_{\text{test}})$ decreases across stages for all methods except Random —, demonstrating successful learning and generalization to unseen environments.

All LLM-generated curricula (active N, N+P, N+P+B, and Static) significantly outperform the Random baseline and demonstrate clear learning. Random — does not show improvement in test performance across stages, suggesting that structured curriculum design (whether from the LLM or expert) is essential for effective learning. LLM-generated curricula — — — — successfully create meaningful learning progressions.

While all LLM methods show learning, none surpass the Expert curriculum in final performance. However, N+P and N+P+B achieve performance statistically equivalent to the expert baseline

(Mann-Whitney U test, $p > 0.05$), indicating that LLM-assisted curriculum generation with appropriate feedback modalities can match expert-level design.

Two variants with feedback outperform Static generation. N+P and N+P+B achieve better test performance than Static (Mann-Whitney U test, all $p < 0.05$). This demonstrates that providing iterative performance feedback enables the LLM to adapt curriculum difficulty and design more effective learning progressions for these versions. These results validate the core hypothesis: active, feedback-informed curriculum generation produces better learning outcomes than static, one-shot curriculum design.

Feedback modality. N+P and N+P+B outperform N. Both progression-augmented (N+P) and behavioral-augmented (N+P+B) feedback yield better test performance than numeric-only feedback (N) (Mann-Whitney U test, $p < 0.05$). The addition of the convergence plot visualization appears to provide the LLM with actionable insights about optimization dynamics, enabling more informed curriculum design (see, e.g., the last stage response of the LLM in the complete interaction shown in Appendix A).

N+P and N+P+B show no significant difference. This suggests that while adding convergence visualization to numeric feedback improves performance, incorporating trajectory visualizations does not provide substantial additional benefits. Analysis of the LLM `understood` and `reasoning` response fields reveals that it correctly interprets the trajectory visualizations and describes observed behaviors accurately. However, this additional information may also introduce prompt complexity without adding decision-relevant insights beyond what convergence plots already provide. The longer, more complex prompts in N+P+B may also introduce cognitive load that offsets potential benefits [34].

4.2.1 Generalization and QD. As we use a QD optimizer (namely, ME), we obtain a set of diverse and good policies at the end of each optimization step, rather than just one good policy. In particular, we use the final gap in x and y axes between the target and the robot position at the end of simulation. Intuitively, different policies in the ME archive are hence diverse in the direction they approach the target from. Indeed, we use these descriptors exactly because they can facilitate the finding of policies that truly generalize the target-reaching behavior beyond the examples provided by training arenas—*i.e.*, to really solve the MT 2D navigation task.

To demonstrate that ME actually exploits this potential in achieving generalization, we show in Figure 5 the performance in the test arenas of the best solution in the ME archive, *i.e.*, the progression and distribution of $\min_s f(s; C_{\text{test}})$.

It can be seen that for all methods there is at least one policy in the archive which scores much better than the best solution chosen based on the performance on the training arenas: e.g., the median $\min_s f(s; C_{\text{test}})$ of the final policy obtained with the N+P+B feedback is ≈ 0.13 (in Figure 5), while the median of $f(s^\star; C_{\text{test}})$ is ≈ 0.22 (in Figure 4). However, the largest difference is in the populations evolved with the Expert curriculum, particularly visible by comparing the progression of the index (—).

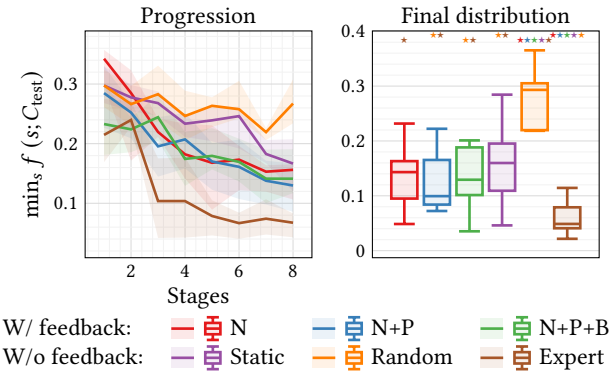

Figure 5: Progression (right plots) and final distribution (left plots) of the performance of the policy $s^\star$ with the best performance on the test arenas, for the three interactive modalities and the three baselines.

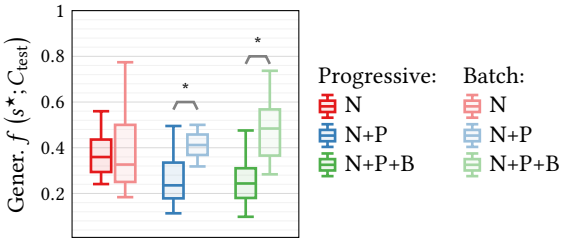

Figure 6: Final distribution of the performance of the best policy $s^\star$ in the population measured on the test arenas for each modality with progressive and batch administration. An arch over a pair of boxes denotes statistically significant difference.

4.3 Progressive vs. batch cases (RQ2)

To verify that the LLM-generated cases form a meaningful curriculum progression rather than arbitrary arena collections, we compare training with progressive administration of cases against training with *batch* administration of all cases. For each of our variant (N, N+P, N+P+B), we take each of the 10 curricula obtained with feedback and perform a single optimization round with a C_{train} containing all the eight curriculum cases. To ensure fair comparison in terms of total episodes, we run the batch training longer, with 40 000 fitness evaluations as stopping criterion: this way we perform the same number of episodes in both conditions ($8 \cdot 40\,000 = \sum_{i=1}^{i=8} 10\,000$).

We show the performance of the obtained policies in Figure 6. The results demonstrate that N+P and N+P+B truly benefit from curriculum structure: progressive training significantly outperforms the batch variants (Mann-Whitney U test, $p < 0.05$). Conversely, there is no difference from progressive and batch administration of curricula obtained with N. This confirms that the augmented feedback provided by progression plots and behavior visualization allows the LLM to generate sequences being genuine curricula, *i.e.*, such that case ordering and gradual accumulation facilitate learning, rather than simply diverse training sets.

4.4 Analyses of LLM responses (RQ3)

Figure 7 illustrates one stage of one interaction loop using the N+P+B modality, where 3 cases have already been generated and evaluated. The LLM receives numeric metrics, convergence plots, and trajectory visualizations, then produces the next case along with explanatory text. The complete sequence of prompts and LLM responses across all stages of this interaction is provided in Appendix A.

By observing the `understood` field of the LLM response in Figure 7, it can be seen that the LLM demonstrates basic competence in interpreting the provided visualizations. It correctly identifies convergence patterns from the progression plots, including the starting quality value, approximate evaluation count at convergence, and relative convergence speed. For trajectory plots, it accurately locates start and target positions and describes the general path shape (e.g., `smooth curve path`, `diagonal path`). The LLM also correctly uses the legend to associate performance curves with specific cases in multi-line plots.

From the `reasoning` field in Figure 7, we see that the LLM is able to exploit the understanding of the feedback for providing and motivating the new case. By looking more broadly at other interactions, we see that when policies achieve low quality scores consistently, the LLM introduces additional obstacles, increases spatial complexity, or varies object positions to maintain challenge progression. Conversely, when performance degrades significantly, it explicitly acknowledges the difficulty spike and generates simpler arenas. Comparing the LLM textual descriptions of intended arena layouts with the actual 2D environments rendered from its character strings reveals reasonable consistency; the generated structures generally match the stated design intentions.

These observations suggest that the LLM can extract relevant information from multimodal feedback and adjust curriculum difficulty accordingly, though its designs represent functional rather than optimal curriculum strategies.

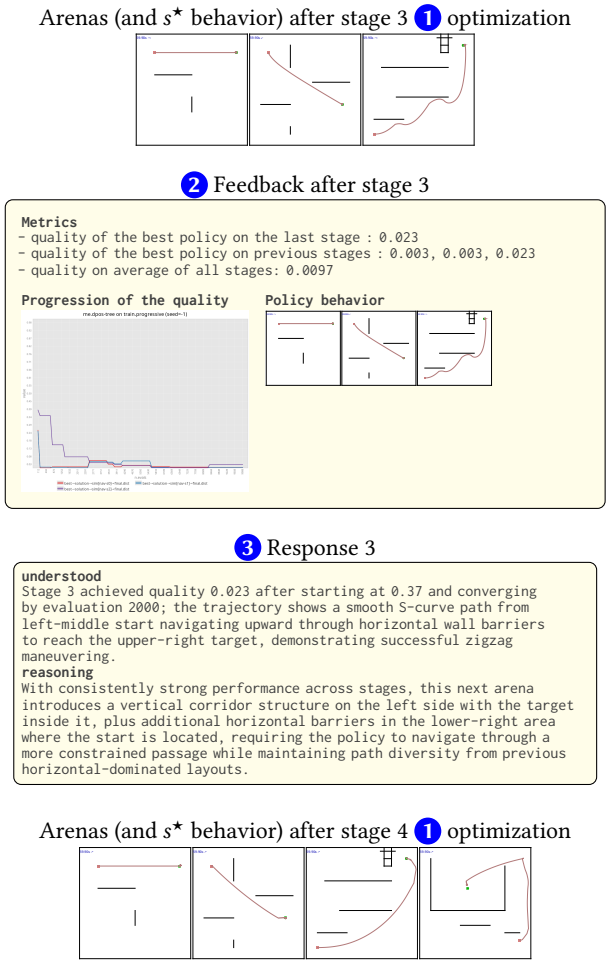

Figure 7: A visualization of a chunk of one run of the interaction with the N+P+B feedback: steps 1 2 3 of stage 3 and step 1 of stage 4. The complete interaction including these stages is shown in Appendix A.

5 Concluding remarks

We introduced an interactive large language model (LLM)-assisted framework that automates curriculum generation for multi-task (MT) evolutionary policy search by leveraging real-time optimization feedback. Experiments on 2D robotic navigation demonstrated that interactive LLM curricula outperform static LLM and random baselines, with progression-augmented feedback (N+P) and behavior-augmented feedback (N+P+B) achieving performance comparable to expert-designed curricula.

Our findings reveal that feedback richness matters: adding convergence plots to numeric metrics improves curriculum quality, though behavioral visualizations offer more modest incremental gains. Analysis of LLM responses confirmed meaningful interpretation of multimodal feedback and adaptive difficulty calibration. Crucially, curricula generated with richer feedback function as genuine learning progressions, as sequential administration outperforms batch training.

This framework is agnostic to optimizer, policy representation, and problem domain, opening pathways for broader evolutionary robotics applications. Future work should investigate scaling to complex robotic tasks, alternative LLMs, and whether incorporating the evolved policies or generalization metrics (e.g., validation set performance) into the feedback loop enables LLMs to better anticipate curriculum impact on unseen cases. This work demonstrates that LLMs can effectively automate curriculum design for embodied artificial intelligence (AI), lowering the expertise barrier for MT learning in evolutionary systems.

Acknowledgments

This publication is based upon work from COST Action Randomised Optimisation Algorithms Research Network (ROAR-NET), CA22137, supported by COST (European Cooperation in Science and Technology).

References

[1] Fuma Aki, Riku Ikeda, Takumi Saito, Ciaran Regan, and Mizuki Oka. 2024. Llm-poet: Evolving complex environments using large language models. In *Proceedings*

of the Genetic and Evolutionary Computation Conference Companion. 243–246.
[2] Anthropic. 2025. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5
[3] Ryan Bahlous-Boldi, Maxence Faldor, Luca Grillotti, Hannah Janmohamed, Lisa Coiffard, Lee Spector, and Antoine Cully. 2025. Dominated novelty search: Rethinking local competition in quality-diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 104–112.
[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.
[5] Ying Bi, Bing Xue, and Mengjie Zhang. 2021. Learning and sharing: A multitask genetic programming approach to image feature learning. *IEEE Transactions on Evolutionary Computation* 26, 2 (2021), 218–232.
[6] Ying Bi, Bing Xue, and Mengjie Zhang. 2022. Multitask feature learning as multiobjective optimization: A new genetic programming approach to image classification. *IEEE Transactions on Cybernetics* 53, 5 (2022), 3007–3020.
[7] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.
[8] Camilo De La Torre, Giorgia Nadizar, Yuri Lavinas, Hervé Luga, Dennis G Wilson, and Sylvain Cussat-Blanc. 2025. Evolution of Inherently Interpretable Visual Control Policies. In *GECCO'25: Genetic and Evolutionary Computation Conference*. ACM, 358–367. doi:10.1145/3712256.3726332
[9] Karol Desnos, Nicolas Sourbier, Pierre-Yves Raumer, Olivier Gesny, and Maxime Pelcat. 2021. Gegelati: Lightweight artificial intelligence through generic and evolvable tangled program graphs. In *Workshop on Design and Architectures for Signal and Image Processing (14th edition)*. 35–43.
[10] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. 2017. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2169–2176.
[11] Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. 2024. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code. *arXiv preprint arXiv:2405.15568* (2024).
[12] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. 2017. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*. PMLR, 482–495.
[13] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.
[14] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. 2017. Automated curriculum learning for neural networks. In *international conference on machine learning*. Pmlr, 1311–1320.
[15] Luca Grillotti and Antoine Cully. 2023. Kheperax: a lightweight jax-based robot control environment for benchmarking quality-diversity algorithms. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 2163–2165.
[16] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. 2019. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3796–3803.
[17] Yuxiao Huang, Wenjie Zhang, Liang Feng, Xingyu Wu, and Kay Chen Tan. 2025. How multimodal integration boost the performance of llm for optimization: Case study on capacitated vehicle routing problems. In *2025 IEEE Symposium for Multidisciplinary Computational Intelligence Incubators (MCII)*. IEEE, 1–7.
[18] Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch. 2008. Multitask visual learning using genetic programming. *Evolutionary computation* 16, 4 (2008), 439–459.
[19] Sébastien Jean, Orhan Firat, and Melvin Johnson. 2019. Adaptive scheduling for multi-task learning. *arXiv preprint arXiv:1909.06434* (2019).
[20] Steven Jorgensen, Giorgia Nadizar, Gloria Pietropolli, Luca Manzoni, Eric Medvet, Una-May O'Reilly, and Erik Hemberg. 2024. Large language model-based test case generation for gp agents. In *Proceedings of the genetic and evolutionary computation conference*. 914–923.
[21] Steven Jorgensen, Giorgia Nadizar, Gloria Pietropolli, Luca Manzoni, Eric Medvet, Una-May O'Reilly, and Erik Hemberg. 2025. Policy Search through Genetic Programming and LLM-assisted Curriculum Learning. *ACM Transactions on Evolutionary Learning* (2025).
[22] Xiaoxu Li, Zifan Ye, Yi Xia, and Ruck Thawonmas. 2025. Dynamic difficulty adjustment using a large language model: A case study in magic: The Gathering. *Entertainment Computing* (2025), 100997.
[23] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. 2024. Environment curriculum generation via large language models. In *8th Annual Conference on Robot Learning*.
[24] Saining Liu, Yi Mei, and Mengjie Zhang. 2025. Curriculum Learning in Genetic Programming Guided Local Search for Large-scale Vehicle Routing Problems. *arXiv preprint arXiv:2505.15839* (2025).
[25] Harald M Ludwig, Ane Espeseth, and Eric Medvet. 2025. Trace-Elites: Better Quality-Diversity with Multi-point Descriptors. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 338–353.
[26] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931* (2023).
[27] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2019. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems* 31, 9 (2019), 3732–3740.
[28] Eric Medvet, Samuele Lippolis, and Giorgia Nadizar. 2025. Quality-diversity in problems with composite solutions: a case study on body–brain robot optimization. *Genetic Programming and Evolvable Machines* 26, 2 (2025), 22.
[29] Eric Medvet, Giorgia Nadizar, and Luca Manzoni. 2022. JGEA: a modular java framework for experimenting with evolutionary computation. In *Proceedings of the genetic and evolutionary computation conference companion*. 2009–2018.
[30] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).
[31] Jean-Baptiste Mouret and Glenn Maguire. 2020. Quality diversity for multi-task optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 121–129.
[32] Giorgia Nadizar, Eric Medvet, and Dennis Wilson. 2024. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 933–941. doi:10.1145/3638529.3653987
[33] Giorgia Nadizar, Eric Medvet, and Dennis G Wilson. 2024. Naturally Interpretable Control Policies via Graph-Based Genetic Programming. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 73–89. doi:10.1145/3643688
[34] Sania Nayab, Giulio Rossolini, Giorgio C Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. 2024. Concise Thoughts: Impact of Output Length on LLM Reasoning and Cost. *CoRR* (2024).
[35] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. 2015. Curriculum learning of multiple tasks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5492–5500.
[36] Kanghyun Ryu, Qiayuan Liao, Zhongyu Li, Payam Delgosha, Koushil Sreenath, and Negar Mehr. 2025. Curricullm: Automatic task curricula design for learning complex robot skills using large language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4470–4477.
[37] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. 2022. Curriculum learning: A survey. *International Journal of Computer Vision* 130, 6 (2022), 1526–1565.
[38] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. 2023. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems* 36 (2023), 54213–54227.
[39] Zhensu Sun, Xiaoning Du, Zhou Yang, Li Li, and David Lo. 2024. Ai coders are among us: Rethinking programming language grammar towards efficient code generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1124–1136.
[40] Quentin Vacher, Stephen Kelly, Ali Naqvi, Nicolas Beuve, Tanya Djavaherpour, Mickaël Dardaillon, and Karol Desnos. 2025. Maple: Multi-action programs through linear evolution for continuous multi-action reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1062–1071.
[41] Neeraj Varshney, Swaroop Mishra, and Chitta Baral. 2022. Let the model decide its curriculum for multitask learning. *arXiv preprint arXiv:2205.09898* (2022).
[42] Linji Wang, Zifan Xu, Peter Stone, and Xuesu Xiao. 2025. GACL: Grounded Adaptive Curriculum Learning with Active Task and Performance Monitoring. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 591–596.
[43] Yanlin Wang, Tianyue Jiang, Mingwei Liu, Jiachi Chen, Mingzhi Mao, Xilin Liu, Yuchi Ma, and Zibin Zheng. 2025. Beyond functional correctness: Investigating coding style inconsistencies in large language models. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 690–712.
[44] Ziping Xu and Ambuj Tewari. 2022. On the statistical benefits of curriculum learning. In *International Conference on Machine Learning*. PMLR, 24663–24682.
[45] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. 2020. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems* 33 (2020), 4767–4777.
[46] Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. 2024. Envgen: Generating and adapting environments via llms for training embodied agents. *arXiv preprint arXiv:2403.12014* (2024).
[47] Xueqiao Zhang, Chao Zhang, Jianwen Sun, Jun Xiao, Yi Yang, and Yawei Luo. 2025. Eduplanner: Llm-based multi-agent systems for customized and intelligent instructional design. *IEEE Transactions on Learning Technologies* (2025).
[48] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering* 34, 12 (2021), 5586–5609.