# Large Language Model-based Test Case Generation for GP Agents

Steven Jorgensen*
MIT CSAIL, MITLL
Cambridge, MA, USA
stevenjson@mit.edu

Giorgia Nadizar*
University of Trieste
Trieste, Italy
giorgia.nadizar@phd.units.it

Gloria Pietropolli*
University of Trieste
Trieste, Italy
gloria.pietropolli@phd.units.it

Luca Manzoni
Eric Medvet
lmanzoni@units.it
emedvet@units.it
University of Trieste
Trieste, Italy

Una-May O'Reilly
Erik Hemberg
unamay@csail.mit.edu
hembergerik@csail.mit.edu
MIT CSAIL
Cambridge, MA, USA

## ABSTRACT

Genetic programming (GP) is a popular problem-solving and optimization technique. However, generating effective test cases for training and evaluating GP programs requires strong domain knowledge. Furthermore, GP programs often prematurely converge on local optima when given excessively difficult problems early in their training. Curriculum learning (CL) has been effective in addressing similar issues across different reinforcement learning (RL) domains, but it requires the manual generation of progressively difficult test cases as well as their careful scheduling. In this work, we leverage the domain knowledge and the strong generative abilities of large language models (LLMs) to generate effective test cases of increasing difficulties and schedule them according to various curricula. We show that by integrating a curriculum scheduler with LLM-generated test cases we can effectively train a GP agent player with environments-based curricula for a single-player game and opponent-based curricula for a multi-player game. Finally, we discuss the benefits and challenges of implementing this method for other problem domains.

## CCS CONCEPTS

• **Computing methodologies → Genetic programming**; • **Information systems → Language models**; • **Software and its engineering** → *Interactive games.*

## KEYWORDS

Linear GP, Large language models, Curriculum learning

**ACM Reference Format:**
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 *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3638529.3654056

## 1 INTRODUCTION

Genetic programming (GP) finds application across multiple domains for tasks ranging from financial trading to civil engineering to binary classification to learning dynamical systems [6, 19, 43, 52, 76]. Its strengths in optimization, global search tasks, and interpretability [44] have also rendered GP effective in the realm of control policies [32, 38, 42] and game playing, where it has been demonstrated to perform competitively on environments such as the Atari set [10, 11, 29, 30, 72] and many others [21, 39, 55].

However, the success or failure of GP often depends on an appropriate ensemble of test cases to optimize upon. As such, test cases need to adequately reflect the complexity of a given problem and need to be relevant and sufficiently challenging. Additionally, GP programs are prone to premature convergence to local sub-optimal solutions, especially when given only overly difficult problems to train on. Thus, crafting effective training and evaluation test cases for GP poses significant challenges. In particular, it requires substantial domain knowledge, especially in those cases when modulating the difficulty of the generated test cases is required.

In fact, the scheduling of test cases at gradually increasing difficulty levels can be used to prevent premature convergence and improve algorithm performance in the long run. Curriculum learning (CL) proposes to incrementally increase problem complexity, facilitating a more natural learning progression by emulating the human learning process. This method proved to be effective in different domains, including robotics [37] and strategy games [54]. Nevertheless, manually creating a curriculum is demanding and strongly depends on the understanding of the problem domain, which can limit the curriculum flexibility and overall effectiveness.

We propose to address some of the aforementioned challenges and shortcomings by leveraging the potentialities of large language models (LLMs), which are known for their advanced natural language processing (NLP) capabilities and content creation [40, 51]. LLMs have recently attracted interest for their application potential across a wide spectrum of fields, including GP and, more broadly, evolutionary computation (EC) [24, 73]. Notably, LLMs have proved effective in guiding the evolution of computer programs [33, 50], confirming their potential to significantly impact GP development.

---

*These authors contributed equally to this research.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

*GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654056

The comprehensive domain knowledge embedded in LLMs, derived from extensive training on diverse datasets, renders this instrument a powerful generative model. For our purpose, LLMs can automatically generate test cases in much larger quantities than would be easily possible by hand. Moreover, thanks to the tunability of LLM output via prompting, it is possible to customize the generation of test cases and strategies, including difficulty tuning, without needing to manually detail each aspect. Along this line, Zala et al. [75] showed their capability at generating adapting environments for reinforcement learning (RL) agents.

In this study, we leverage the power of LLMs to create test cases, which we schedule according to a curriculum, to train GP agents. To assess our method, which remains general and potentially applicable to a wide range of scenarios, we focus on the challenging domain of games. We test our approach on two well-known games—Connect-4, a two-player board game, and Super Mario Bros, a single-player video game—evolving a GP agent player with the linear genetic programming (LGP) representation. While the former provides a simpler, two-player setting for initial method evaluation, the latter presents a more complex, single-player challenge, offering a comprehensive proving ground for our techniques. Moreover, as these games require different types of test cases, namely opponent player policies for Connect-4 and game environments for Super Mario Bros, they allow us to showcase the generality of our method.

In summary, our contribution is twofold: (1) we demonstrate the use of LLMs for generating diverse test cases for optimizing GP agents; (2) we investigate effective test case scheduling methods for enhancing GP learning, evaluating the impact of CL. Our findings confirm the effectiveness of LLMs at creating relevant test cases for diverse problem settings. Moreover, we show how CL can induce the development of more robust solutions compared to optimizing solely on challenging test cases in both game scenarios.

## 2 BACKGROUND

### 2.1 Linear genetic programming

LGP is a variant of GP where programs are represented as lists of instructions in an assembly-like language which manipulate the content of a predefined set of registers. [7, 27].

Previous studies have utilized LGP for optimizing controllers in various settings. Videau et al. [69] have applied it, together with standard tree-based GP, for solving several discrete and continuous control tasks. Similarly, Nadizar et al. [42] tested LGP along with Cartesian GP on a set of continuous control tasks. Kelly et al. [31] have also tackled a continuous control problem with LGP, exploiting its potential for evolving computer programs to find a robust control program for a multi-legged robot.

In a LGP program each instruction is defined by specifying (1) the index of the register where the result of the execution will be stored, (2) the function to execute from those available in the function set $H$, and (3) the sequence of arguments to be passed to the function $(i_1, \dots, i_{m_{\mathrm{ar}}})$, expressed in terms of register indexes ($m_{\mathrm{ar}} = 2$ being the maximum arity for functions in $H$). To represent a program of $n_{\mathrm{lines}}$ lines, the genome is expressed as a sequence of bounded integers of size $(2 + m_{\mathrm{ar}})n_{\mathrm{lines}}$. The bounds for register indexes are determined by the amount of available registers $n_{\mathrm{reg}}$, whereas the ones for functions are derived by the size $|H|$ of the function set. At the start of the execution, the inputs of the program are copied into the first $n_{\mathrm{in}}$ registers, while the outputs are extracted from the last $n_{\mathrm{out}}$ registers after the execution of the last instruction. All other registers are initialized to 0 to avoid assignment issues.

### 2.2 Curriculum learning

CL, introduced by Elman [16] but formally defined by Bengio et al. [5], is a training strategy that mimics the progressive learning pattern observed in animals and humans [71]. This approach is based on the concept of "start small" and organizes the learning in a curriculum (i.e., a sequence of tasks) where the task difficulty is gradually increased. Different studies confirmed that CL can provide performance improvements over a standard training approach and accelerate the training process, without imposing additional computational costs [58]. Thanks to these advantages, CL has found widespread application in multiple areas such as computer vision [23, 26], natural language processing [49, 61], robot manipulation tasks [41], RL tasks [17, 45, 46], and many more.

### 2.3 Large language models

LLMs are probabilistic models of natural language trained on vast datasets of text, usually for general-purpose language generation [78]. This makes them quite useful for NLP tasks such as the summarization of articles [36], text classification [9], or generation of fake reviews [3]. These models are based on a neural network architecture known as a *transformer*, introduced by Devlin et al. [14] and further improved by Vaswani et al. [68], which utilizes a concept known as model "attention" to focus the model on the important parts of the input. The introduction of OpenAI's "GPT" model series boosted the visibility of LLMs tremendously, especially since their release of "ChatGPT" due to the free, simple-to-use interface for question answering [8, 47].

To interact with an LLM, users generally provide a text query known as *prompt*. The prompt conveys the context, meaning, and intention of the user to the LLM. The prompt is tokenized and fed to the LLM which outputs a sequence of text, with the intention of generating the ideal completion to the prompt. All LLMs have a finite amount of tokens they can accept as input and have no memory of previous prompts unless aggregated into the new prompt. They also do not have a guarantee of correctness for their answers and have been shown to sometimes return "hallucinated" facts [77].

## 3 METHOD

Here we provide an overview of our method. First, we create a prompt detailing the high-level specifications and, optionally, the target difficulty for the test cases we want to generate. We then feed the prompt to the LLM to generate the requested test case. We optionally iterate this procedure to generate multiple cases; modulating the specifications and/or the difficulty and relying on the LLM stochasticity for achieving variety. We then schedule the generated test cases into a *curriculum*, which we define as a sequence of pairs $(\langle T_i, b_i \rangle)_i$, where $T_i = \{t_j\}_j$ is a set of test cases and $b_i$ is a computational budget. We optimize a GP agent according to a curriculum by iterating over the sequence $(\langle T_i, b_i \rangle)_i$ and for each $i = 1, \dots, n$ we assess the population on $T_i$ until exhausting the budget $b_i$ before proceeding to $i + 1$. If $|T_i| > 1$, we define a suitable strategy for exploiting all test cases in $T_i$. To adhere to the more classical notion of CL, we group test cases according to their difficulty, i.e., each $T_i$ is associated to a certain difficulty, and we create sequences of increasing difficulties only.

We deploy our general method in two game environments—a single-player game and a two-player game—and exploit LLMs to generate test cases for optimizing a GP agent as player. For the single-player game, the test cases constitute the game environments whereas for the two-player game, they are the opponent players. We further detail our approach in the following sections.

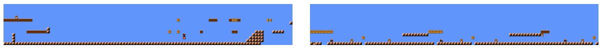
## 3.1 Game environments

*Connect-4.* First, we consider *Connect-4*, a popular two-player board game, that presents complex decision-making challenges for an agent to solve and is characterized by the intrinsic difficulty that each move influences the game outcome, which remains until the conclusion of the game [12]. This game has been widely used as a benchmark for assessing the performance of game-playing agents. A popular approach for creating effective Connect-4 agents is RL [20, 62], although various other strategies have been adopted over years [53, 56, 59]. However, to the best of our knowledge, no GP-based agent has been specifically developed for this task, except for a preliminary study [1].

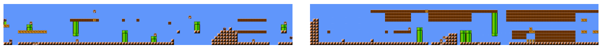
Connect-4 is a two-player game on a $7 \times 6$ board. The first player (yellow) and the second player (red) alternatively drop a piece into one of the columns, and the piece settles at the lowest available space. The objective for each player is to align four pieces horizontally, vertically, or diagonally. The game has a moderately complex state space, with around $4.5 \times 10^{12}$ possible board configurations [15]. While other games, like chess or Go, provide a larger search space [2], we selected Connect-4 as a reasonable middle ground between the complexity of the task and ease of analysis. For this study, we will use a simplified $6 \times 6$ board.

*Super Mario Bros.* For the single-player game, we consider *Super Mario Bros*, a classic video game that was released by Nintendo in 1985. Thanks to its potentially high-dimensional input space, tunable difficulty, and strategy requirements for achieving good scores, this video game has inspired the creation of an RL benchmark [66] along with several competitions based upon it [28]. This ignited researchers' interest in developing agents to solve the problem environment, yielding various contributions over the years [65, 67]. Among them, two contributions have encompassed GP: Perez et al. [48] relied on grammatical evolution for optimizing controllers in the form of behavior trees, whereas de Freitas et al. [13] utilized grammar-based GP to evolve a program controller.

The game-play of Super Mario Bros involves navigating an agent—Mario—through different *levels* (see Figure 1), overcoming obstacles such as blocks and pipes, and defeating or avoiding enemies. We refer to each level as a *test case* in the following sections. A test case is considered solved if the agent is able to fully traverse it from left to right. To this end, the agent can perform five actions at every game time-step: (1) move left, (2) move right, (3) move down, (4) jump, and (5) run (which only has effect if combined with a move action). We rely on the game framework and simulator originally developed for the MarioAI competition by Togelius et al. [66] for our evaluation.



**(a) Trivial: "little blocks, no enemies, no pipes".** **(b) Easy: "little blocks, some enemies, no pipes".**



**(c) Medium: "little blocks, some enemies, some pipes".** **(d) Difficult: "many blocks, many enemies, many pipes".**

**Figure 1: Sample of LLM-generated test cases for Super Mario Bros for each difficulty, with corresponding prompts.**

## 3.2 GP agents

For both considered games, we evolve a GP agent as a player, using the LGP representation.

*Connect-4.* For Connect-4, we evolve a GP agent to learn the optimal column for placing a piece on the board. We train a GP agent by having it compete against one or more predefined opponent policies, aiming for it to outplay them. At each time step of the game, we query the GP agent for its decision, providing as input the current state of the board and expecting the chosen column as output. Therefore the *observation* consists of the current state of the board, encoded as an integer vector of length $n_{\text{in}} = 36$. We obtain the vector by flattening a matrix representation of the board, where 0 indicates an empty cell, 1 a piece placed by the GP agent, and $-1$ a piece placed by the opponent. Regarding the GP agent's *actions*, we consider the outputs from its final $n_{\text{out}}$ registers, where $n_{\text{out}} = 36$ is the dimension of the grid. For determining the action, i.e., the board column where to put the piece, we take the index of the largest value among output registers modulo the number of columns (i.e., 6). To measure the quality of the GP agent, we define a fitness measure based on the accumulated rewards $r$ obtained during the game, defined as the sum $r = \sum_{i=1}^{n_{\text{moves}}} r_i$ of partial rewards $r$, where $n_{\text{moves}}$ is the number of moves performed before the game ends. We define the partial reward for each move as follows. A winning move gives a positive reward of $r_i = 10$, while a losing move gives a negative ($r_i = -1$) one. When there is no winner, aligning three or two pieces gives the GP agent a reward of 0.6 and 0.3 respectively. Instead, when the opponent aligns three pieces the reward is negative ($r_i = -0.5$). If the agent makes an illegal move (such as placing a piece in a column that is already full) the game ends and it receives a negative reward ($r_i = -10$). In all other cases, the reward is 0.

*Super Mario Bros.* For Super Mario Bros, we evolve the GP agent to navigate test cases. We query the GP agent at every game time-step, providing it with a (lossy) encoding of the game scenario and expecting it to output the actions to be taken. In the *observation* we encode the world surrounding Mario as an integer vector to be fed to the GP agent as follows. First, we discretize the world from pixel granularity to element granularity, where elements are blocks, pipes, and enemies. Next, we reduce the observation size to consider only the $8 \times 8$ matrix centered around the agent. We then encode blocks and pipes as 1, enemies as $-1$, and empty spaces as 0. Finally, we flatten the integer matrix into a vector of length $n_{\text{in}} = 64$. Concerning the *actions* to be taken, we consider the output of the last $n_{\text{out}} = 5$ registers. For each of these registers, if it exceeds a threshold of 0 we trigger the corresponding action.

If two contrasting actions are triggered, e.g., move left and move right, we delegate the conflict resolution to the game simulator. To measure the quality of the GP agent, we consider the degree to which it is able to solve a test case. Namely, we assign the traversed percentage, computed along the $x$-axis, of the test case as *fitness*, which we aim to maximize.

### 3.3 LLMs for test case generation

Using an LLM to generate test cases presents distinct challenges for the two considered games, as it needs to produce intrinsically different outputs. For Connect-4 a test case consists of an opponent policy, whereas for Super Mario Bros it consists of a game environment (i.e., a level).

*Connect-4.* To generate test cases, i.e., opponent policies, for the Connect-4 GP agent we use ChatGPT-4 [47]. Before proceeding further into the discussion, it is important to clarify that existing policies for training agents to play Connect-4 are available and relatively straightforward for developers to implement. However, we aim to demonstrate that an LLM like ChatGPT-4 can effectively generate policies that enable a GP agent to learn. To generate effective policies of different difficulties, we began by communicating to the LLM our requirement for diverse opponent strategies in Connect-4 (see the Appendix). At this point, the LLM proposed and explained different strategies varying in complexity.

We generate: (1) an *easy* policy, focusing on immediate advantageous moves without long-term planning, which helps an agent learn elementary tactics; (2) a *medium* policy, which is similar to a greedy approach but also has a 50 % chance of blocking an opponent's winning moves; (3) a *difficult* policy, which is an advanced version of the greedy strategy that always blocks the opponent's potential winning moves. We note that the LLM also provided the policy implementation (in a specific programming language, provided as input by the user). We embedded the provided code within our framework with minor post-processing needed to align them with the game simulation interface.

*Super Mario Bros.* For generating the test cases for Super Mario Bros we rely on *MarioGPT* [60], a fine-tuned GPT2 model trained to generate tile-based game levels. The main characteristic of MarioGPT is that it can be text-prompted for controlling the generation of new game levels. However, unlike most off-the-shelf LLMs, MarioGPT only accepts prompts represented as combinations of specific features ("blocks", "pipes", and "enemies") alongside quantitative keywords ("no", "little", "some", "many"). Within the prompt it is also possible to specify the level elevation—low or high—yet for simplicity we disregard that aspect in our evaluation. The generated test cases are reported to reflect the prompts with different accuracies for each element [60], namely 92 % for blocks, 68 % for enemies, and 81 % for pipes.

MarioGPT represents the generated levels, i.e., the test cases, as strings where each character encodes for a particular game element, e.g., different types of enemies, blocks, and pipes. The test case representation is compatible with the MarioAI simulator format, meaning that no intermediate processing is needed. As reported by Sudhakaran et al. [60], 88 % of the generated test cases are playable, i.e., there exists a path that allows its traversal. Before using a generated test case for training a LGP agent we test its feasibility, as suggested by the authors of MarioGPT, by running Robin Baumgarten's A* agent [66] on it. If it is not solvable, we repeat the generation, decreasing the temperature to reduce the stochasticity, until a playable level is found.

Given the constrained nature of the prompts, we rely on four prompts that correspond to four increasing difficulties. We consider: (1) *trivial* test cases with "little blocks, no enemies, no pipes", where the agent should learn the basic game play; (2) *easy* test cases with "little blocks, some enemies, no pipes" where the agent should learn to deal with enemies; (3) *medium* test cases with "little blocks, some enemies, some pipes", where the agent should learn to overcome taller and wider obstacles as the pipes in addition to dealing with enemies; and (4) *difficult* test cases with "many blocks, many enemies, many pipes", where the agent should learn to face a complex game scenario. We report an example of test case for each difficulty in Figure 1. It is worth noting that the prompt-test case correspondence is less than perfect, as sometimes the generation does not reflect the request. We remark that for each difficulty we can generate a potentially infinite number of test cases for training the GP agent with little to no human effort, simply by prompting MarioGPT and relying on its stochasticity.

Similar to Connect-4, Super Mario Bros presents an existing set of available test cases deriving from both the original game and the MarioAI competition [66]. However, given our goal of relying on an LLM only for generating test cases, we neglect their existence in the optimization phase and only employ them as validation cases.

### 3.4 Scheduling test cases

Given the LLM-generated test cases, an additional degree of freedom to explore is how to schedule them to achieve successful GP agent training.

*Connect-4.* For Connect-4, we have three LLM-generated policies of increasing difficulty. Thus, a natural approach is to organize them in a curriculum, $(\langle T_i, b_i \rangle)_i$, with $i = 1, 2, 3$, where $T_1 = \{t_e\}$ consists only of the easy policy $t_e$, $T_2 = \{t_m\}$ consists only of the medium policy $t_m$, and $T_3 = \{t_d\}$ consists only of the difficult policy $t_d$.

The relative simplicity of Connect-4 (especially compared to our other case study, Super Mario Bros) allows us to explore various curriculum variants by fixing the number $n_{\text{gen}}$ of generations for the evolution of the LGP population and distributing these generations across different opponent policies, with each $b_i$ indicating the percentage of generations consumed on the $i$-th policy. For brevity, we denote by $b_1$-$b_2$-$b_3$ a curriculum $(\langle \{t_e\}, b_1 \rangle, \langle \{t_m\}, b_2 \rangle, \langle \{t_d\}, b_3 \rangle)$—when $b_i = 0$, the corresponding $i$-th pair is absent from the curriculum. We also evolve a GP agent solely against the difficult policy, represented as 0-0-100, to compare the impact of the curriculum approach.

*Super Mario Bros.* Concerning Super Mario Bros, we consider test cases belonging to four difficulties. Therefore, as for Connect-4, we can also exploit CL for optimizing the GP agent. We define a curriculum with four increasing difficulties $(\langle T_i, b_i \rangle)_i$, with $i = 1, \dots, 4$, together with a baseline consisting only of the last stage, i.e., of the most difficult one, $T_4$. We recall that for Super Mario Bros the LLM—MarioGPT—can generate a potentially infinite amount of test cases per prompt, i.e., per difficulty, thanks to its intrinsic stochasticity. This enables us to consider multiple (namely, $n_t$) test cases for each difficulty, i.e., $\forall i, |T_i| = n_t$, naturally fostering the generalization ability of the agent and accounting for the imperfect prompt-test case matching of MarioGPT. We consider two approaches for dealing with multiple test cases of the same difficulty: a *parallel* one, where we assess each agent on all the test cases $t \in T_i$ and average the performance across them, and a *sequential* one, where we test individuals on one test case $t \in T_i$ at a time, cyclically replacing it if we detect the stagnation of evolution for $n_{\text{stag}}$ generations. In summary, we consider the following curricula for the Super Mario Bros: (1) *C–PAR*, a CL schedule $(\langle T_i, b_i \rangle)_i$, with $i = 1, \dots, 4$, $b_1 = \dots = b_4 = 25$, and parallel test case evaluation at each $T_i$; (2) *C-SEQ*, a CL schedule $(\langle T_i, b_i \rangle)_i$, with $i = 1, \dots, 4$, $b_1 = \dots = b_4 = 25$, and sequential test case evaluation at each $T_i$; (3) *D-PAR*, the C-PAR version considering only $T_4$, i.e., $b_1 = \dots = b_3 = 0$ and $b_4 = 100$; and (4) *D-SEQ*, the C-SEQ version considering only $T_4$, i.e., $b_1 = \dots = b_3 = 0$ and $b_4 = 100$. For each of these, we equally divide the total computational budget among the stages of the curriculum.

## 4 EXPERIMENTS AND RESULTS

We perform a thorough experimental evaluation to assess our proposed method, which we detail in the following. In particular, we focus on addressing the following research questions:

RQ1 Can we exploit LLMs to generate test cases for training a GP agent?
RQ2 Given LLM-generated test cases, how can we schedule the training to best leverage them? Is CL a viable strategy?

### 4.1 Parameters and procedure

For optimizing the GP agents, we rely on a mutation-only genetic algorithm (GA) with elitism. We evolve a population of size $n_{\text{pop}} = 50$ for Connect-4 and $n_{\text{pop}} = 100$ for Super Mario Bros for $n_{\text{gen}}$ generations. To determine $n_{\text{gen}}$, we fix the total amount of fitness evaluations $n_{\text{evals}}$ regardless of the curriculum employed: we set $n_{\text{evals}} = 5 \times 10^3$ for Connect-4 and $n_{\text{evals}} = 4 \times 10^5$ for Super Mario Bros. For the generation of each genome, we sample a uniform distribution over the allowed integer values for each position. At each generation, we select $n_{\text{par}} = 0.9 n_{\text{pop}}$ parent genomes relying on tournament selection, with size $n_{\text{tour}} = 3$. From each of these, we generate an offspring genome using int-flip mutations. We assign a different mutation probability to the left-hand side of each program line, i.e., the registers to be assigned, ($p_a = 0.3$), to the functions ($p_f = 0.1$), and to the function inputs ($p_i = 0.1$). We obtain the new population by merging the offspring with the elite from the previous generation; consisting of the best $n_{\text{elite}} = 0.1 n_{\text{pop}}$ genomes.

Concerning the representation, we recall we employ LGP. We consider the following function set $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \sqrt{\bullet}^*, \bullet < \bullet, \bullet > \bullet, \bullet < 0, \bullet > 0\}$, where $\bullet$ represents an operand, and operators marked with $^*$ are protected from undefined behavior. The last four functions are Boolean functions, where the output is 1 if the condition is satisfied and 0 otherwise. We set $n_{\text{reg}} = n_{\text{in}} + n_{\text{const}} + n_{\text{out}} + 5$ and $n_{\text{lines}} = 20$. The addition of $n_{\text{const}}$ stems from the fact that we add two constant inputs to each observation, namely $\{0.1, 1\}$.

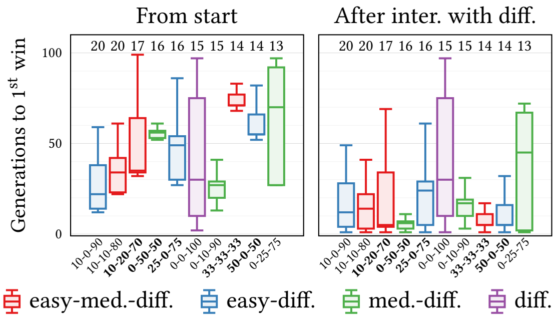


**Figure 2: Distribution of the generation for the first victory of the best GP agent against the difficult policy, across 20 runs. On the left we report the distribution over the total number of generations, on the right the distribution counted from the first encounter with the difficult policy. We use bold for the $x$-labels relative to the curricula differing in a statistically significant way from the non-curriculum strategy (0-0-100).**

Concerning the curricula, we consider the scenarios described in Section 3.4. For Connect-4 we experiment with a total of 10 combinations of budget allocation among stages. For Super Mario Bros, we set $n_t$, i.e., the number of test cases per difficulty, to 3 as a reasonable trade-off between computational cost and test cases variability. As for the stagnation parameter for C-SEQ and D-SEQ, we set $n_{\text{stag}} = 10$.

After evolving the GP agents, we further measure their abilities by assessing them on test cases not seen during evolution. For Connect-4, we test the agent using a strategy it has not been trained against, i.e. the *minimax* with a search depth of 3 moves, a well-known approach that involves predicting and negating the opponent's moves for optimal play. For Super Mario Bros we consider 8 test cases not seen during evolution generated by MarioGPT along with 15 test cases extracted from the original Super Mario Bros game.

To obtain statistically significant results, we perform 20 independent runs of each configuration for Connect-4 and 10 independent runs for Super Mario Bros. When comparing samples, we test statistical significance with a Wilcoxon rank-sum test with equality as null hypothesis and $\alpha = 0.05$. We release our results and our code for allowing experiments reproducibility[1].

### 4.2 Results

*Connect-4.* To assess the effectiveness of the training policies generated by the LLM and to determine if the GP agents are learning to play, we initially evaluate their ability to win a game. The left plot of Figure 2 reports the earliest generation in which the GP agent wins against the difficult policy across 20 runs, using different curricula, including the baseline comparison where training is solely against the difficult policy (0-0-100). This analysis includes only those runs where the best agent wins against the difficult policy within the evolutionary timeline. The number of such runs is reported above each plot, and the boxplots are sorted in descending order of win rate.

---

[1] https://github.com/gpietrop/LLM-Connect4
https://github.com/giorgia-nadizar/MarioGP-T

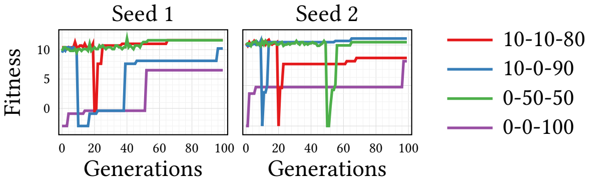
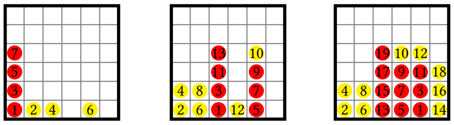


**Figure 3: Fitness of the best GP agent across generations for two different seeds.**



**(a) Easy.** **(b) Medium.** **(c) Difficult.**

**Figure 4: Best GP agent (in red) in the 10-0-90 configuration playing Connect-4 against different policies. We decorate the boards by adding the move number on top of each piece.**

Considering that using a curriculum approach implies that the agent plays against the difficult policy only after some rounds against the easy and/or the medium policy, the right plot of Figure 2 also presents an alternative version of the data. We chose this presentation to enable a clear understanding of when the agent achieves its first victory against the difficult policy, starting the count from the agent's first interaction with it. The right plot in Figure 2 shows that after the initial curriculum training, agents rapidly reach a victory outcome.

These results answer RQ1 positively and demonstrate the GP agent's capability to learn effectively from LLM-generated policies across various curricula, including the case where a curriculum is not adopted, ensuring success against the difficult policy. Furthermore, they answer RQ2: certain curricula improve the agent's learning efficiency and victory rate over difficult-only curriculum (0-0-100). For example, while the 0-0-100 agent wins over the difficult policy 15 out of 20 times, two other curricula (10-0-90 and 10-10-80) consistently win. These findings suggest that extensive training against easier policies is not required to boost performance; a brief initial session with a weaker opponent before moving to a more challenging one is more effective. Additionally, structuring the curriculum with some interactions with the easy policy appears to be more beneficial w.r.t. starting directly with a medium one.

Results also indicate that the baseline 0-0-100 tends to win games earlier in evolution when it does succeed. It is important to highlight that these agents only compete against the difficult policy, offering them a chance of success across all generations, unlike the other curricula. To assess if the number of generations to the first win under various curricula differs from the baseline (0-0-100), we employ the Wilcoxon rank-sum test. We highlight statistically significant differences by bolding the $x$-labels in Figure 2.

To investigate the learning progress of different curricula, we report in Figure 3 the progression of the fitness of the best GP agent, for two seeds. We decided to report results on individual seeds rather than aggregate values, as learning behavior shows significant variation across seeds, and analyzing individual cases is more informative. We report the evolution of the most successful curriculum for each category and, as a baseline, include the performance of the agent trained only against the difficult policy.

These results give a hint on how a curriculum-trained agent reacts to a change of policy. These seeds show all agents win, but the fitness score is different as a consequence of our definition of fitness, which rewards wins and penalizes for opponent alignments. A lower score for the baseline agent compared to others like the 10-10-80 agent indicates better defense against opponent strategies. Moreover, these results show an immediate success of curriculum-trained agents against the easy and medium policies they are initially trained on, despite oscillations due to the opponent's stochasticity. Thereafter, in the transition to the difficult policy, there is an initial phase of fitness loss. However, after a few generations, these agents (nearly always) return to winning, demonstrating rapid adaptation.

Finally, in Figure 4 we report a game played by the best GP agent (10-0-90), tested against the three policies used in this study. Red pieces are the agent's, yellow are the opponent's. The evolved agent consistently wins, illustrating its ability to adapt and win against various strategies. Figure 4 also offers a visual representation of the increasing difficulty of different LLM-generated policies. The first game board shows a relatively straightforward win for our agent, reflecting a policy designed to teach basic moves. The medium policy is more challenging, yet the agent wins as it is not consistently blocked. The final game, against the difficult policy, is the most interesting one as it demonstrates the agent's ability to develop a strategic approach by creating multiple paths to victory.

We also test our method against the minimax strategy, known for its optimal play. In 10 games each, both the baseline (0-0-100) and best curriculum (10-0-90) agent win 3 times. The results are not surprising given that minimax is an advanced strategy compared to the ones used against the agent during the training. We recall that our goal is to demonstrate the LLM effectiveness in creating test cases and the potential advantages of scheduling them according to CL, not to develop an infallible policy. Winning against minimax even a few times signals the potential of our approach.

*Super Mario Bros.* To evaluate whether the LLM-generated test cases are effective for GP agent training in Super Mario Bros, we report in Figure 5 (left) the progression of the average test case completion (averaged across the $n_t = 3$ considered test cases per difficulty) achieved by the best individual in the population along fitness evaluations. For both C-PAR and D-PAR, the value reported corresponds to the fitness, whereas for C-SEQ and D-SEQ the metrics are computed separately from the fitness, every 50 generations (i.e., every 5 000 fitness evaluations). In fact, for the latter cases, the fitness is evaluated on a single test case per difficulty at a time, switching in case of stagnation every $n_{\text{stag}} = 10$ generations. However, this results in an unstable fitness, which would make the plot noisy and unreadable. We decorate the plot with vertical separators that indicate the points during evolution at which we change the difficulty for the CL cases, C-PAR and C-SEQ.

From this plot we can gain several insights. First, we can confirm the occurrence of learning for the parallel scheduling cases, C-PAR and D-PAR, as indicated by an increase in fitness over time. Concerning the sequential scheduling cases, instead, the lines for both C-SEQ and D-SEQ appear noisy and unstable, without a neat

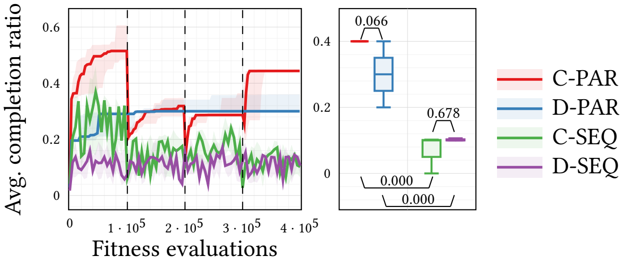


**Figure 5: Progression and final distributions of the average completion fraction (averaged across $n_t = 3$ test cases) achieved collected by the best GP agent in the population at each iteration; median and inter-quartile range across 10 independent runs. For C-PAR and C-SEQ the vertical dashed lines indicate the change of difficulty in the curriculum (trivial-easy-medium-difficult); D-PAR and D-SEQ are always evaluated against difficult test cases. We decorate pairs of distributions with the $p$-values resulting from pairwise Wilcoxon rank-sum tests with equality as null hypothesis.**

growing trend, meaning that learning is hindered by this scheduling. This might hint that the considered test cases for any difficulty are different enough from one another to promote the survival and reproduction of different individuals, which tend to become "temporary specialists" on any test case under exam, succumbing to others whenever the test case changes. Conversely, relying on a parallel curriculum fosters the emergence of generalists that simultaneously gain expertise on the features of every test case. Hence, based on this analysis, we can positively answer RQ1 and conclude that LLM-generated test cases can be effectively employed for GP agent training, given that an appropriate curriculum is chosen. Namely, scheduling the evaluation on multiple test cases in parallel outperforms sequential scheduling.

Another interesting analysis concerns the comparison between the 4-difficulties curricula and the difficult-only curricula. Within the sequential scheduling, there seem to be no relevant differences, as both C-SEQ and D-SEQ lead to poor evolutionary performance.

Conversely, examining the parallel scheduling, the differences between C-PAR and D-PAR are worth highlighting. Focusing on C-PAR, we notice the typical trend of CL: for each considered difficulty there is learning, followed by a drop in performance at every change of scenario. Remarkably, these drops are not total, i.e., the average test case completion does not go to 0, meaning that some ability is retained, yet the performance naturally decreases because of the newly introduced hurdles. However, there is an almost immediate growth thereafter, as the population quickly learns to overcome the new obstacles. Examining D-PAR, instead, we do notice a rather fast evolution at the beginning, which is followed by a plateau, meaning the population has converged to some local optimum. Thus, from these results, we can positively answer RQ2; CL has enabled evolution to escape local optima and achieve a final better performance by gradually increasing test case difficulty.

To further compare the results with different scheduling schemes, we also display the distribution of the average completion fraction achieved by the best individual in the population at the end of evolution in Figure 5 (right). We decorate pairs of boxplots with the $p$-value resulting from a pairwise statistical test. These results compliment the previous analysis, allowing us to also examine the statistical relevance of the observed differences. Namely, the $p$-values confirm both the indistinguishable nature of C-SEQ and D-SEQ, and the neat superiority of the parallel scheduling w.r.t. its sequential counterpart. However, the $p$-value resulting from the comparison of C-PAR and D-PAR does not highlight statistical significance. Hence, in conclusion C-PAR seems to be the best option, yet with no statistical relevance. We remark, though, that these results are obtained from 10 independent runs only, due to the high computational costs (each run takes roughly 24 h on a Xeon-p8 48 core machine), and the results could possibly gain statistical significance with more executions.

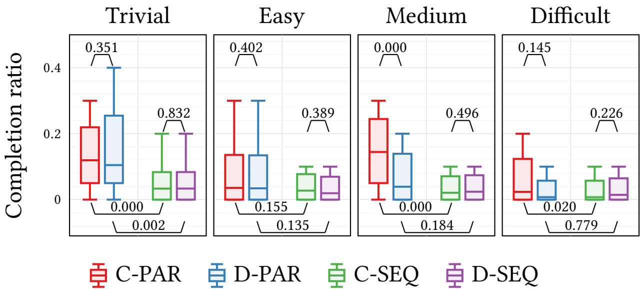


**Figure 6: Distribution of the completion fraction achieved by the 40 best LGP agents (10 seeds, 4 curricula) on 8 new LLM-generated test cases per difficulty. We decorate pairs of distributions with the $p$-values resulting from pairwise Wilcoxon rank-sum tests with equality as null hypothesis.**

Proceeding to the generalization results, we display the distribution of the performance obtained by the best evolved individuals on the 8 new LLM-generated test cases, grouped by difficulty (see the titles of the plots), in Figure 6. As for the previous figure, we decorate pairs of boxplots with $p$-values resulting from statistical analyses. Testing the agents on unseen test cases is fundamental to assess if they overfit or they actually learned the game play.

In Figure 6 we analyze the results according to the difficulty of the tasks. Focusing on trivial and easy test cases, we note that for both parallel and sequential scheduling, CL does not play a significant role. This finding aligns with previous CL works [18, 70], which show that agents that focus on harder tasks only are still able to obtain performance benefits when evaluated on easier versions of the task. Concerning the differences between parallel and sequential scheduling, we highlight the superiority of parallel scheduling only on trivial cases. This partially contradicts previous findings, and indicates that sequential scheduling can foster generalization ability in some cases. Moving to medium difficulty, we observe how C-PAR truly shines against all other scheduling curricula, which all follow behind. In this case, being trained in parallel and on multiple difficulties has enabled the agents to gain better generalization abilities. Last, for difficult test cases we note no statistical differences. In this scenario, it is expected that all agents behave reasonably well, as they have all encountered these types of test cases in training.

To conclude our analysis, we show the evolved agents performance on the preexisting test cases in Table 1. Namely, we report the rank achieved, in median, by the best individual evolved according to each curriculum—C-PAR, D-PAR, C-SEQ, and D-SEQ—for each test case (one per column). We do not detail the absolute

**Table 1: Ranking of each curriculum on 15 original Super Mario Bros levels. Rank 1 and 4 correspond to the best and worst performing cases, respectively. We assign the lower rank to both in case of ties.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | Sum |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C-PAR | 2 | **1** | **1** | **1** | **1** | 2 | **1** | **1** | **1** | 2 | 2 | **1** | 2 | **1** | **1** | 20 |
| D-PAR | 3 | 2 | 2 | 2 | 3 | **1** | 3 | 2 | 3 | **1** | **1** | 3 | **1** | 2 | 2 | 31 |
| C-SEQ | **1** | 4 | 3 | 4 | 3 | 3 | 3 | 4 | **1** | 3 | 3 | 3 | 4 | 4 | 3 | 46 |
| D-SEQ | 4 | 3 | 4 | 2 | 2 | 4 | 3 | 3 | 4 | 4 | 4 | 3 | 3 | 2 | 3 | 48 |

performance obtained as it is line w.r.t. that achieved on the LLM-generated test cases depicted in Figure 6. From this table we can again reason comparatively among the considered curricula, observing that C-PAR generally outperforms all other curricula, always ranking first or second, confirming most of our previous findings. Following right behind there is D-PAR, usually ranked second or third, whereas both sequential curricula fall neatly behind, with remarkably worse rankings. These results are in line with our overall findings, confirming the utility of LLM-generated test cases for training GP agents. Moreover, they affirm the importance of relying on multiple test cases in parallel, if available, arranging them into a curriculum of increasing difficulty.

To conclude our analysis, we show videos of one GP agent per curriculum solving one of the difficult LLM-generated test cases, at https://giorgia-nadizar.github.io/MarioGP-T/. From these we can remark again the superiority of the parallel curricula w.r.t. the sequential ones, as in both C-SEQ and D-SEQ the agent immediately succumbs to an obstacle, whereas in C-PAR and D-PAR it almost reaches the conclusion of the test case.

### 4.3 Discussion

The analysis in Section 4.2 indicates that LLM-generated test cases enable GP agents to learn effectively in both two-player and single-player games. GP agents win against LLM-generated strategies in the two-player game and progress through LLM-generated levels in the single-player game, positively addressing RQ1. Additionally, the integration of a curriculum scheduler into the training of GP agents often improves performance over a non-scheduled training approach. For simpler games like Connect-4, with low computational requirements, we investigated different curriculum budget allocations. Interestingly, Connect-4 did not require extensive training on easier levels; a brief focus was enough to improve the robustness of the agent as well as the success rate. Conversely, a more complex game like Super Mario Bros provided more test cases of equal difficulty; allowing for the development of diverse strategies to exploit them. Parallel scheduling results highlight the impact of CL in a single-player game, leading to better performance on test cases encountered during the training and showing improved adaptability to new situations. These observations address our second research question RQ2, confirming the effectiveness of CL when integrated with LLM-generated test cases.

## 5 CONCLUSION, LIMITATIONS, AND FUTURE WORK

We devised an approach for exploiting large language models (LLMs) for generating test cases for the training of genetic programming (GP) agents. Moreover, we proposed to schedule LLM-generated test cases according to the principles of curriculum learning (CL), to maximize their effectiveness. We validated our approach against two challenging game scenarios, a two-player board game, and a single-player video game, proving not only the feasibility of the method but also its effectiveness. In addition, we showed the importance of gradually increasing the difficulty of the test cases during learning for achieving well-performing and robust GP agents.

While our methods yield promising results, there are key considerations and challenges in using LLMs for test case generation. Firstly, the effectiveness of LLM-generated cases relies on crafting precise *prompts*, a task that becomes crucial with general LLMs like ChatGPT-4. Guo et al. [22] suggest evolving prompts to improve this process. However, the inherent stochasticity of LLMs can cause the same prompt to lead to different outcomes, hindering the reproducibility [4]. Another point to consider is the *familiarity* of the LLM with the problem domain and potential biases from its training [25, 74]. For unfamiliar domains, breaking them into known segments or providing wider context can help the LLM generate more effective outputs [57]. Moreover, MarioGPT and similar studies, such as [64], demonstrate that fine-tuning LLMs can also be effective in tackling the lack of domain knowledge. However, as noted by Li et al. [35], this approach demands considerable computational effort and training data.

Future research should assess the generality of our approach by exploring other GP commonly used in control tasks and video games, like Cartesian GP [72] and Tangled Program Graphs [29], while also examining diverse CL strategies and test case scheduling [71]. In addition, gaining insights from this preliminary analysis, our approach could also be tested on more complex multi-player games like Othello [34], leveraging our findings on fitness budget allocations and on the benefits of relying on multiple test cases in parallel, when available. For advanced single-player scenarios, it could be worth exploring Sokoban, for which Todd et al. [64] developed a fine-tuned LLMs for level generation. Last, building on top of our proposition, another direction involves automatically improving prompt design for better test case generation. Introducing an evolutionary-based prompt optimization [23] could enable a co-evolutionary setup [63], allowing prompts and agents to evolve together to improve the overall performance.

## ACKNOWLEDGMENTS

This research is partially supported by the PRIN 2022 PNRR project "Cellular Automata Synthesis for Cryptography Applications (CASCA)" (P2022MPFRT) financed by the European Union – Next Generation EU.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Department of the Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Air Force.

## REFERENCES

[1] Mohamed F. Abdelsadek. [n. d.]. Using Genetic Programming to Evolve a Connect-4 game player.

[2] Louis Victor Allis et al. 1994. *Searching for solutions in games and artificial intelligence.* Ponsen & Looijen Wageningen.

[3] Alberto Bartoli and Eric Medvet. 2020. Exploring the Potential of GPT-2 for Generating Fake Reviews of Research Papers. In *Fuzzy Systems and Data Mining VI*. IOS Press, 390–396.

[4] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 610–623.

[5] 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.

[6] Anthony Brabazon, Michael Kampouridis, and Michael O'Neill. 2020. Applications of genetic programming to finance and economics: past, present, future. *Genetic Programming and Evolvable Machines* 21 (2020), 33–53.

[7] Markus Brameier, Wolfgang Banzhaf, and Wolfgang Banzhaf. 2007. *Linear genetic programming*. Vol. 1. Springer.

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

[9] Youngjin Chae and Thomas Davidson. 2023. Large language models for text classification: From zero-shot learning to fine-tuning. *Open Science Foundation* (2023).

[10] Tim Cofala, Lars Elend, and Oliver Kramer. 2020. Tournament Selection Improves Cartesian Genetic Programming for Atari Games.. In *ESANN*. 345–350.

[11] Leonardo Lucio Custode and Giovanni Iacca. 2022. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 224–227.

[12] Mayank Dabas, Nishthavan Dahiya, and Pratish Pushparaj. 2022. Solving Connect 4 Using Artificial Intelligence. In *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2021, Volume 1*. Springer, 727–735.

[13] João Marcos de Freitas, Felipe Rafael de Souza, and Heder S Bernardino. 2018. Evolving controllers for mario AI using grammar-based genetic programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[15] Stefan Edelkamp and Peter Kissmann. 2008. Symbolic classification of general two-player games. In *KI 2008: Advances in Artificial Intelligence: 31st Annual German Conference on AI, KI 2008, Kaiserslautern, Germany, September 23-26, 2008. Proceedings 31*. Springer, 185–192.

[16] Jeffrey L Elman. 1993. Learning and development in neural networks: The importance of starting small. *Cognition* 48, 1 (1993), 71–99.

[17] 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.

[18] Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner SchmidSiemens AG, et al. 2019. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 898–906.

[19] Sébastien Gaucel, Maarten Keijzer, Evelyne Lutton, and Alberto Tonda. 2014. Learning dynamical systems using standard symbolic regression. In *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*. Springer, 25–36.

[20] Imran Ghory. 2004. Reinforcement learning in board games. *Department of Computer Science, University of Bristol, Tech. Rep* 105 (2004).

[21] Robert Gold, Henrique Branquinho, Erik Hemberg, Una-May O'Reilly, and Pablo García-Sánchez. 2023. Genetic Programming and Coevolution to Play the Bomberman™ Video Game. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 765–779.

[22] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532* (2023).

[23] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R Scott, and Dinglong Huang. 2018. Curriculumnet: Weakly supervised learning from large-scale web images. In *Proceedings of the European conference on computer vision (ECCV)*. 135–150.

[24] Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. 2024. Evolving Code with A Large Language Model. https://api.semanticscholar.org/CorpusID:266999468

[25] Dong Huang, Qingwen Bu, Jie Zhang, Xiaofei Xie, Junjie Chen, and Heming Cui. 2023. Bias assessment and mitigation in llm-based code generation. *arXiv preprint arXiv:2309.14345* (2023).

[26] Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G Hauptmann. 2014. Easy samples first: Self-paced reranking for zero-example multimedia search. In *Proceedings of the 22nd ACM international conference on Multimedia*. 547–556.

[27] Wolfgang Kantschik and Wolfgang Banzhaf. 2002. Linear-graph GP-a new GP structure. In *European Conference on Genetic Programming*. Springer, 83–92.

[28] Sergey Karakovskiy and Julian Togelius. 2012. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 55–67.

[29] Stephen Kelly and Malcolm I Heywood. 2017. Emergent tangled graph representations for Atari game playing agents. In *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20*. Springer, 64–79.

[30] Stephen Kelly and Malcolm I Heywood. 2017. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 195–202.

[31] Stephen Kelly, Daniel S Park, Xingyou Song, Mitchell McIntire, Pranav Nashikkar, Ritam Guha, Wolfgang Banzhaf, Kalyanmoy Deb, Vishnu Naresh Boddeti, Jie Tan, et al. 2023. Discovering Adaptable Symbolic Algorithms from Scratch. *arXiv preprint arXiv:2307.16890* (2023).

[32] Stephen Kelly, Tatiana Voegerl, Wolfgang Banzhaf, and Cedric Gondro. 2021. Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genetic Programming and Evolvable Machines* 22 (2021), 573–605.

[33] Joel Lehman, Jonathan Gordon, Shawn Jain, Cathy Yeh, Kenneth Stanley, and Kamal Ndousse. 2024. *Evolution Through Large Models*. 331–366. https://doi.org/10.1007/978-981-99-3814-8_11

[34] Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2022. Emergent world representations: Exploring a sequence model trained on a synthetic task. *arXiv preprint arXiv:2210.13382* (2022).

[35] Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, Steve Jiang, and You Zhang. 2023. ChatDoctor: A Medical Chat Model Fine-Tuned on a Large Language Model Meta-AI (LLaMA) Using Medical Domain Knowledge. *Cureus* 15, 6 (2023).

[36] Yixin Liu, Alexander R Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. 2023. On Learning to Summarize with Large Language Models as References. *arXiv preprint arXiv:2305.14239* (2023).

[37] Sha Luo, Hamidreza Kasaei, and Lambert Schomaker. 2020. Accelerating reinforcement learning for reaching using continuous curriculum learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[38] Francesco Marchetti, Gloria Pietropolli, Federico Julian Camerota Verdù, Mauro Castelli, and Edmondo Minisci. [n. d.]. Control Law Automatic Design Through Parametrized Genetic Programming with Adjoint State Method Gradient Evaluation. *Available at SSRN 4490005* ([n. d.]).

[39] Giovanna Martinez-Arellano, Richard Cant, and David Woods. 2016. Creating AI characters for fighting games using genetic programming. *IEEE transactions on computational intelligence and Ai in games* 9, 4 (2016), 423–434.

[40] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[41] Adithyavairavan Murali, Lerrel Pinto, Dhiraj Gandhi, and Abhinav Gupta. 2018. Cassl: Curriculum accelerated self-supervised learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6453–6460.

[42] 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.

[43] Giorgia Nadizar and Gloria Pietropolli. 2023. A grammatical evolution approach to the automatic inference of P systems. *Journal of Membrane Computing* 5, 3 (2023), 129–143.

[44] Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. 2024. An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-Loop and Genetic Programming. *ACM Trans. Evol. Learn. Optim.* (2024).

[45] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research* 21, 1 (2020), 7382–7431.

[46] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. 2017. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning.. In *IJCAI*. 2536–2542.

[47] R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article* 2 (2023), 13.

[48] Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. 2011. Evolving behaviour trees for the mario ai competition using grammatical evolution. In *European Conference on the Applications of Evolutionary Computation*. Springer, 123–132.

[49] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. 2019. Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848* (2019).

[50] 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. 2023. Mathematical discoveries from program search with large language models. *Nature* (2023), 1–3.

[51] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems* 35 (2022), 36479–36494.

[52] Leo Willyanto Santoso, Bhopendra Singh, S Suman Rajest, Rajan Regin, and Karrar Hameed Kadhim. 2021. A genetic programming approach to binary classification problem. *EAI Endorsed Transactions on Energy Web* 8, 31 (2021), e11–e11.

[53] Marvin Oliver Schneider and JL Garcia Rosa. 2002. Neural connect 4-A connectionist approach to the game. In *VII Brazilian Symposium on Neural Networks, 2002. SBRN 2002. Proceedings.* IEEE, 236–241.

[54] Kun Shao, Yuanheng Zhu, and Dongbin Zhao. 2018. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence* 3, 1 (2018), 73–84.

[55] Yehonatan Shichel, Eran Ziserman, and Moshe Sipper. 2005. GP-robocode: Using genetic programming to evolve robocode players. In *European Conference on Genetic Programming.* Springer, 143–154.

[56] Peer Sommerlund. 1996. Artificial neural nets applied to strategic games. *Unpublished, last access* 5 (1996), 12.

[57] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2998–3009.

[58] 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.

[59] Martin Stenmark. 2005. Synthesizing board evaluation functions for Connect-4 using machine learning techniques. *Master's thesis, Østfold University College, Norway* (2005).

[60] S Sudhakaran, M González-Duque, C Glanois, M Freiberger, E Najarro, and S Risi. [n. d.]. MarioGPT: open-ended text2level generation through large language models (2023). *arXiv preprint arxiv:2302.05981* ([n. d.]).

[61] Yi Tay, Shuohang Wang, Luu Anh Tuan, Jie Fu, Minh C Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. 2019. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. *arXiv preprint arXiv:1905.10847* (2019).

[62] Markus Thill, Patrick Koch, and Wolfgang Konen. 2012. Reinforcement learning with n-tuples on the game Connect-4. In *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12.* Springer, 184–194.

[63] John N Thompson. 2014. *Interaction and coevolution.* University of Chicago Press.

[64] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games.* 1–8.

[65] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. 2010. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation.* IEEE, 1–8.

[66] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jurgen Schmidhuber. 2009. Super mario evolution. In *2009 ieee symposium on computational intelligence and games.* IEEE, 156–161.

[67] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis. 2013. The mario ai championship 2009-2012. *AI Magazine* 34, 3 (2013), 89–92.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[69] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. 2022. Multi-objective genetic programming for explainable reinforcement learning. In *European Conference on Genetic Programming (Part of EvoStar).* Springer, 278–293.

[70] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference.* 142–151.

[71] Xin Wang, Yudong Chen, and Wenwu Zhu. 2021. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 9 (2021), 4555–4576.

[72] Dennis G Wilson, Sylvain Cussat-Blanc, Hervé Luga, and Julian F Miller. 2018. Evolving simple programs for playing Atari games. In *Proceedings of the genetic and evolutionary computation conference.* 229–236.

[73] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. 2024. Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap. *arXiv preprint arXiv:2401.10034* (2024).

[74] Kai-Ching Yeh, Jou-An Chi, Da-Chen Lian, and Shu-Kai Hsieh. 2023. Evaluating Interfaced LLM Bias. In *Proceedings of the 35th Conference on Computational Linguistics and Speech Processing (ROCLING 2023).* 292–299.

[75] Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. 2024. EnvGen: Generating and Adapting Environments via LLMs for Training Embodied Agents. arXiv:2403.12014 [cs.CL]

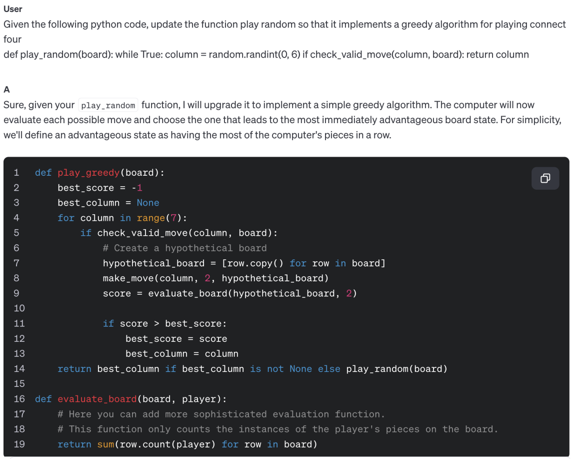
[76] Qianyun Zhang, Kaveh Barri, Pengcheng Jiao, Hadi Salehi, and Amir H Alavi. 2021. Genetic programming in civil engineering: Advent, applications and future trends. *Artificial Intelligence Review* 54 (2021), 1863–1885.

[77] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).

[78] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

# Appendix: examples of Connect-4 LLM prompts and results

An example of a prompt given to ChatGPT-4 to generate a Connect-4 policy for the GP agents to train on is shown in Figure 1. The LLM is able to generate a Python implementation of a greedy Connect-4 algorithm based on the random Python policy it was prompted with.



**User**

Given the following python code, update the function play random so that it implements a greedy algorithm for playing connect four

def play_random(board): while True: column = random.randint(0, 6) if check_valid_move(column, board): return column

**A**

Sure, given your `play_random` function, I will upgrade it to implement a simple greedy algorithm. The computer will now evaluate each possible move and choose the one that leads to the most immediately advantageous board state. For simplicity, we'll define an advantageous state as having the most of the computer's pieces in a row.

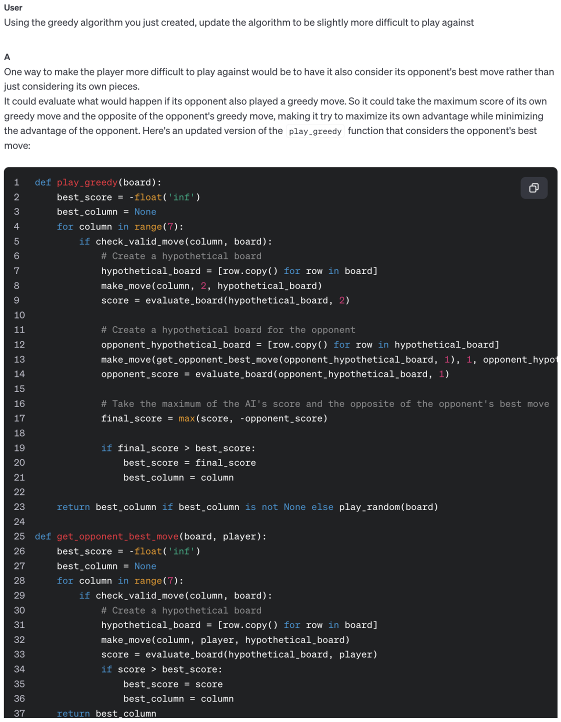
```
def play_greedy(board):
    best_score = -1
    best_column = None
    for column in range(7):
        if check_valid_move(column, board):
            # Create a hypothetical board
            hypothetical_board = [row.copy() for row in board]
            make_move(column, 2, hypothetical_board)
            score = evaluate_board(hypothetical_board, 2)

            if score > best_score:
                best_score = score
                best_column = column
    return best_column if best_column is not None else play_random(board)

def evaluate_board(board, player):
    # Here you can add more sophisticated evaluation function.
    # This function only counts the instances of the player's pieces on the board.
    return sum(row.count(player) for row in board)
```

**Figure 1: ChatGPT-4 example prompt and output for a Connect-4 opponent policy.**

Figure 2 demonstrates the LLM updating the previously produced greedy policy to make it more difficult to play against by adding the capability to consider the opponents next move. Note that due to the stochastic nature of LLMs, this policy is slightly different than some of the policies we generated in our experiments.



**User**

Using the greedy algorithm you just created, update the algorithm to be slightly more difficult to play against

**A**

One way to make the player more difficult to play against would be to have it also consider its opponent's best move rather than just considering its own pieces.

It could evaluate what would happen if its opponent also played a greedy move. So it could take the maximum score of its own greedy move and the opposite of the opponent's greedy move, making it try to maximize its own advantage while minimizing the advantage of the opponent. Here's an updated version of the `play_greedy` function that considers the opponent's best move:

```
def play_greedy(board):
    best_score = -float('inf')
    best_column = None
    for column in range(7):
        if check_valid_move(column, board):
            # Create a hypothetical board
            hypothetical_board = [row.copy() for row in board]
            make_move(column, 2, hypothetical_board)
            score = evaluate_board(hypothetical_board, 2)

            # Create a hypothetical board for the opponent
            opponent_hypothetical_board = [row.copy() for row in hypothetical_board]
            make_move(get_opponent_best_move(opponent_hypothetical_board, 1), 1, opponent_hypot
            opponent_score = evaluate_board(opponent_hypothetical_board, 1)

            # Take the maximum of the AI's score and the opposite of the opponent's best move
            final_score = max(score, -opponent_score)

            if final_score > best_score:
                best_score = final_score
                best_column = column

    return best_column if best_column is not None else play_random(board)

def get_opponent_best_move(board, player):
    best_score = -float('inf')
    best_column = None
    for column in range(7):
        if check_valid_move(column, board):
            # Create a hypothetical board
            hypothetical_board = [row.copy() for row in board]
            make_move(column, player, hypothetical_board)
            score = evaluate_board(hypothetical_board, player)
            if score > best_score:
                best_score = score
                best_column = column
    return best_column
```

**Figure 2: ChatGPT-4 example prompt and output for an updated version of the greedy Connect-4 opponent policy seen in Figure 1.**