Policy Search through Genetic Programming and LLM-assisted Curriculum Learning

STEVEN JORGENSEN, MIT CSAIL, Cambridge, Massachusetts, USA
GIORGIA NADIZAR, Dipartimento di Ingegneria e Architettura, University of Trieste, Trieste, Italy
GLORIA PIETROPOLLI and LUCA MANZONI, Dipartimento di Matematica, Informatica e Geoscienze, University of Trieste, Trieste, Italy
ERIC MEDVET, Dipartimento di Ingegneria e Architettura, University of Trieste, Trieste, Italy
UNA-MAY O'REILLY and ERIK HEMBERG, MIT CSAIL, Cambridge, Massachusetts, USA

Curriculum learning (CL) consists in using a diverse set of user-provided test cases, with varying levels of difficulty and organized in a suitable progression, for learning a policy. The quality of test cases is important to allow optimization techniques as genetic programming (GP) to solve policy search problems. In this work, we evaluate large language models (LLMs) as providers of test cases for GP-based policy search. We consider two policy search tasks, a single-player and a multi-player game, and four LLMs differing in complexity and specialization, which we prompt in order to generate suitable test cases for the two games. We experimentally assess the intrinsic quality of LLM-generated test cases and their utility when inserted in a curriculum consumed by a GP optimization. We evaluate the robustness of the approach with respect to the way cases are scheduled in curricula and with respect to the policy representation, for which we use both graphs and linear programs evolved by GP. We observe that the effectiveness of LLM-assisted CL depends on both the choice of LLM and the design of the prompting and scheduling strategies. These findings highlight important considerations for leveraging LLMs in automated curriculum design for GP-based optimization.

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

Additional Key Words and Phrases: Graph-based GP, Large language models, Curriculum learning

Steven Jorgensen, Giorgia Nadizar, and Gloria Pietropolli contributed equally to this research.

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

Authors' Contact Information: Steven Jorgensen, MIT CSAIL, Cambridge, Massachusetts, USA; e-mail: stevenjson@mit.edu; Giorgia Nadizar, Dipartimento di Ingegneria e Architettura, University of Trieste, Trieste, Italy; e-mail: giorgia.nadizar@dia.units.it; Gloria Pietropolli, Dipartimento di Matematica, Informatica e Geoscienze, University of Trieste, Trieste, Italy; e-mail: gloria.pietropolli@units.it; Luca Manzoni, Dipartimento di Matematica, Informatica e Geoscienze, University of Trieste, Trieste, Italy; e-mail: lmanzoni@units.it; Eric Medvet (corresponding author), Dipartimento di Ingegneria e Architettura, University of Trieste, Trieste, Italy; e-mail: emedvet@units.it; Una-May O'Reilly, MIT CSAIL, Cambridge, Massachusetts, USA; e-mail: unamay@csail.mit.edu; Erik Hemberg, MIT CSAIL, Cambridge, Massachusetts, USA; e-mail: hembergerik@csail.mit.edu.

ACM 2688-3007/2026/6-ART14
https://doi.org/10.1145/3772718

ACM Reference format:
Steven Jorgensen, Giorgia Nadizar, Gloria Pietropolli, Luca Manzoni, Eric Medvet, Una-May O'Reilly, and Erik Hemberg. 2026. Policy Search through Genetic Programming and LLM-assisted Curriculum Learning. *ACM Trans. Evol. Learn. Optim.* 6, 2, Article 14 (June 2026), 37 pages.
https://doi.org/10.1145/3772718

1 Introduction

Due to its adaptability to a wide range of search spaces, **genetic programming (GP)** has emerged as a versatile optimization technique [De Lorenzo et al., 2020]. Its successful applications span diverse domains, including financial trading [Brabazon et al., 2020], civil engineering [Zhang et al., 2021], and symbolic dynamical systems [Leporati et al., 2023; Nadizar and Pietropolli, 2023]. When used with the proper solution representation, GP can be used to optimize programs which are transparent, hence enabling interpretability [Nadizar et al., 2024c], and effective. Many recent research works showed that GP is particularly capable of evolving programs which govern the behavior of agents, i.e., policies, for control [Kelly et al., 2021; Marchetti et al., 2024; Nadizar et al., 2024b] or video-game playing [Cofala et al., 2020; Custode and Iacca, 2022; Gold et al., 2023; Kelly and Heywood, 2017a,b; Shichel et al., 2005; Wilson et al., 2018a].

In general, the success or failure of GP often depends on an appropriate ensemble of test cases to guide optimization. Test cases need to broadly cover the domain of the problem, be relevant and sufficiently challenging. Failing to meet these requirements would make GP search ineffectual, leading to solutions which are not general or not good enough to solve problem instances of realistic difficulty. Ideally, it would be beneficial to present test cases following an increasing level of difficulty, such that the optimization always faces the proper level of challenge. Presenting test cases to the learning or optimization algorithm with gradually increasing difficulty levels is called **curriculum learning (CL)**. CL aims at facilitating a smoother learning progression by emulating the human learning process. It has been proven to be effective in different domains, including robotics [Luo et al., 2020] and strategy games [Shao et al., 2018]. Nevertheless, manually creating a curriculum is demanding and has a strong dependence on understanding the problem domain, which can limit the curriculum flexibility and overall effectiveness. Even when incrementally difficult test cases are available, it remains an open question as to how to schedule them—a bit like deciding how to schedule the development of artificial agents [Nadizar et al., 2022]. How long should a level of difficulty be used? Should mixed levels of difficulty be used? Should test cases within a level be presented concurrently or is sequential presentation better?

We propose to address some of the aforementioned challenges by leveraging **large language models (LLMs)**, which serve not only as advanced tools for **natural language processing (NLP)** content creation [Min et al., 2023; Saharia et al., 2022], but also as foundation models that incorporate extensive knowledge across a wide range of domains. It could be possible to explain a problem to a LLM without needing to manually detail each aspect and successfully task it to provide incrementally more difficult test cases. This may depend on the competence of a specific LLM and prompt engineering. It may require a prompt with in-context learning or even a series of prompts. LLMs have recently attracted interest for their application potential across a wide spectrum of fields, including GP and, more broadly, **evolutionary computation (EC)** [Hemberg et al., 2024; Wu et al., 2024]. Notably, LLMs have proved effective in guiding the evolution of computer programs [Lehman et al., 2024; Romera-Paredes et al., 2024], confirming their potential to significantly impact GP development. 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, and without requiring the user to focus on low level implementation details. Along this line, Zala et al. [2024] showed their capability at generating adapting environments for **reinforcement learning (RL)** agents. An LLM-based approach may also offer representational flexibility. An LLM can generate game configurations of different levels of difficulty, which can serve as test cases for a game-agent evolved with GP. It could also generate code to play as an opponent in a two-player game to provide feedback to a game-agent.

In this work, we leverage the power of LLMs to create progressively more difficult test cases for a single-player game and a two-player game with game levels and generated opponent policies that increase in complexity and playing ability. We schedule the test cases according to a curriculum and present them to GP which searches for a policy for the game-agent. 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, provides a simpler, two-player setting for initial method evaluation. Test cases are presented to Connect-4 in the form of opponent policies. The other game, Super Mario Bros, is a single-player video game that is open-ended and progressively more challenging, offering a more comprehensive proving ground for our method: here, test cases are presented as game levels. For both games, we evolve game-agent policies with GP, but we resort to two different representations for the policies: graphs and linear programs. Consequentially, we use **cartesian genetic programming (CGP)** and **linear genetic programming (LGP)**, two forms of **graph-based genetic programming (GGP)**. Because these games require different types of test cases, namely opponent policies for Connect-4 and game levels for Super Mario Bros, they allow us to showcase the generality of our method.

Our contribution is manifold: (1) we demonstrate the use of LLMs for generating progressively more difficult test cases for optimizing game-agent policies through GP; (2) we investigate effective scheduling methods for enhancing GP learning, evaluating the impact of CL; (3) we investigate the effect that the policy representation has on GP search effectiveness when evolving on the generated test cases; and (4) we compare LLMs of different complexity and specialization in their ability to generate test cases. Our work provides a systematic evaluation of how different LLMs and prompting strategies affect the quality and diversity of generated test cases for GP-based policy search. We observe that the ability to control test case difficulty and the effectiveness of prompting strategies differ substantially across LLMs. Furthermore, our experiments show that CL, when driven by LLM-generated test cases, can lead to more robust solutions compared to training exclusively on the most challenging instances, across multiple game scenarios. This work extends [Jorgensen et al., 2024], where we presented the preliminary results of this study. We broaden the analysis of the initial work by evaluating the robustness of the approach with respect to the variant of GP, including both LGP and CGP, and to the specific LLM being employed. We also deepen the analysis by providing more results and extended discussion.

We proceed as follows: Section 2 provides background on GGP, Connect-4, Super Mario Bros, and LLMs. Section 2 also covers related work. Section 3 presents our method. Section 4.2 presents experiments and evaluation. Section 5 concludes.

2 Background and Related Work

2.1 GGP

Although GP originated as a method for evolving computer programs represented internally as trees [Koza, 1994], graph-based alternatives have gained significant traction over the years [Françoso Dal Piccol Sotto et al., 2021]. In this work, we consider two GGP variants, LGP [Kantschik and Banzhaf, 2002] and CGP [Miller and Thomson, 2000], which share a linear integer genotype and a

phenotype interpretable as a **directed acyclic graph (DAG)**. These approaches are discussed in detail in Sections 2.1.1 and Section 2.1.2.

2.1.1 LGP. LGP is a variant of GGP where programs are represented as lists of instructions in an assembly-like language which manipulate the content of a predefined set of registers [Brameier and Banzhaf, 2007; Kantschik and Banzhaf, 2002].

In an 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 $(i_1, \ldots, i_{m_{\text{ar}}})$, expressed in terms of register indexes ($m_{\text{ar}} = 2$ being the maximum arity for functions in our H). To represent a program of n_{lines} lines, the genotype is expressed as a sequence of bounded integers of size $(2 + m_{\text{ar}})n_{\text{lines}}$. The bounds for register indexes are determined by the amount of available registers n_{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_{in} registers, while the outputs are extracted from the last n_{out} registers after the execution of the last instruction. All other registers are initialized to 0 to avoid assignment issues.

2.1.2 CGP. CGP encodes programs as grids of nodes arranged in a Cartesian plane [Miller, 2020; Miller and Thomson, 2000]. Each node represents a function, which can take as arguments either the outputs of nodes from preceding layers or the inputs of the program. The final program outputs are derived from the outputs of n_{out} selected nodes.

In this work, we simplify the representation to a 1D grid, i.e., a sequence of nodes with a total length of n_{nodes}. Fully specifying a CGP graph thus requires: (1) n_{out} indices to define the program outputs, and (2) n_{nodes} tuples $(h, i_1, \ldots, i_{m_{\text{ar}}})$, where each tuple defines a node function and its arguments. Here, m_{ar} represents the maximum arity of the functions in the set H, which is 2 in our case. Each tuple is constrained as follows: h is an index limited by the size of H, and each argument index can range from 0 to $n_{\text{in}} + j - 1$, where j is the current node position in the sequence. Consequently, the genotype of a CGP graph is represented as a sequence of bounded integers with a total size of $n_{\text{out}} + (1 + m_{\text{ar}})n_{\text{nodes}}$.

2.1.3 Related Work on GGP. Various previous studies have utilized GGP for optimizing controllers in various settings, mostly focusing on the interpretability of the resulting policies. Wilson et al. [2018a] relied on mixed-type CGP for finding simple and well-performing policies for playing the Atari games [Machado et al., 2018], while De La Torre et al. [2025] more recently leveraged a multi-type extension of CGP [De La Torre et al., 2024] for the same task. Wilson et al. [2018b] also proposed a positional variant of CGP where the position of nodes was subject to evolution, and tested it on nine benchmark problems, including three control problems.

Videau et al. [2022] applied LGP, together with standard tree-based GP, for solving several discrete and continuous control tasks. Similarly, Nadizar et al. [2024b] tested both LGP and CGP on a set of continuous control tasks. Later, the same authors leveraged quality-diversity optimization algorithms to obtain CGP policies differing both in terms of agent behavior and graph structure [Nadizar et al., 2024a, 2025]. Kelly et al. [2023] 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.

Another noteworthy GGP representation is **tangled program graphs (TPGs)**, which, while not employed in this work, is closely related to our approaches. Kelly and Heywood [2017a] first utilized TPGs to solve Atari games, even creating a single program capable of solving multiple games [Kelly and Heywood, 2017b; Kelly et al., 2021]. More recently, Amaral et al. [2022] demonstrated the potential of hybridizing TPGs with Symbiotic Bid-based GP for a continuous locomotion control

task. Finally, Vacher et al. [2025] extended TPGs using multiple LGPs to address multi-action continuous control tasks.

2.2 CL

Scaffolding is an element of curriculum design in human education. It is a structured approach of teaching progressively more difficult material. It sequences content so that each lesson builds upon the previous one. Starting with basic concepts, more sophisticated concepts are introduced as a learner demonstrates readiness, often through successfully completing a formative assessment (often a test). CL is a machine learning technical approach to model training that mimics scaffolding. As such, it 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 [Wang et al., 2021]. It was first introduced by Elman [1993] and then formally defined by Bengio et al. [2009].

Related Work on CL. Different studies confirmed that CL can provide performance improvements over a standard training approach and accelerate the training process, without imposing additional computational costs [Soviany et al., 2022]. Thanks to these advantages, CL has found widespread application in multiple areas such as computer vision [Guo et al., 2018; Jiang et al., 2014], NLP [Platanios et al., 2019; Tay et al., 2019], self-supervised learning [Murali et al., 2018], RL [Florensa et al., 2017; Narvekar et al., 2017, 2020], and many more.

CL has also been effectively used for the progressive complexification of learning environments. The **paired open-ended trailblazer (POET)** algorithm [Wang et al., 2019] exemplifies this by autonomously generating increasingly complex environments while co-evolving agents to solve them. POET generalizes CL by automating and diversifying the curriculum-building process, enabling open-ended discovery and robust skill acquisition beyond what fixed or linear curricula can achieve. A related approach is presented by Milano and Nolfi [2021], which enhances evolutionary training of embodied agents by integrating a CL mechanism that automatically selects environmental conditions for evaluation. Their method outperforms standard training and yields robust, adaptable solutions—similar to our findings. Building on the ideas behind POET, the recent OMNI-EPIC framework [Faldor et al., 2024] uses large foundation models, such as advanced LLMs, to autonomously generate novel environments and their associated reward functions as executable code. OMNI-EPIC further automates and extends CL by leveraging LLMs to generate, select, and sequence tasks based on agent progress, fostering continual and open-ended development of increasingly complex behaviors. This aligns closely with our approach, where we also use LLMs to generate diverse test cases as learning environments.

Another automated CL approach uses transfer learning. Weinshall et al. [2018] pre-train a teacher model to estimate the difficulty of each training example, enabling a scheduler to sequence the curriculum accordingly. Dynamic curricula can also be developed using "data parameters," where each data point or class is assigned a learnable weight that influences its impact during training [Saxena et al., 2019]. These weights are updated alongside model parameters, automatically prioritizing examples based on their learning value.

2.3 LLMs

LLMs are probabilistic models of natural language trained on vast datasets of text, usually for general-purpose language generation [Zhao et al., 2023]. This makes them quite useful for NLP tasks such as the summarization of articles [Liu et al., 2024], text classification [Chae and Davidson, 2023], or generation of fake reviews [Bartoli and Medvet, 2020]. These models are based on a neural network architecture known as a *transformer*, introduced by Vaswani et al. [2017], which utilizes a concept known as model "attention" to focus the model on the important parts of the input. The

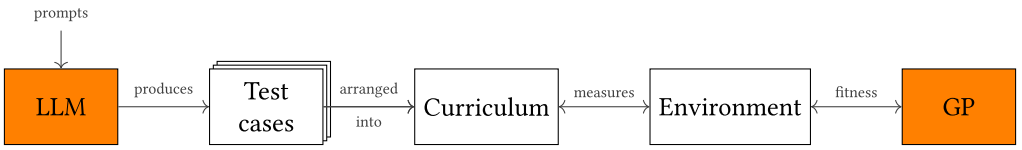


Fig. 1. Overview of the proposed method.

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 [Achiam et al., 2023; Brown et al., 2020].

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 [Zhang et al., 2023].

Related Work on LLMs and GP. There is a growing body of work that integrates LLMs with GP and, more in general, with EC [He et al., 2025; Wu et al., 2024]. Hemberg et al. [2024] identifies several papers integrating LLMs with GP, including using LLMs to initialize the GP population, employing them as mutation and crossover operators, and even utilizing LLMs to evaluate the fitness of GP agents. To date, no GP work exists in enlisting an LLM for test case generation.

3 Method

Here, we provide an overview of our method (see Figure 1). 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 $i = 1, \ldots, n$ we assess the population on T_i until exhausting the budget b_i before proceeding to $i + 1$. If $|T_i| > 1$, we implement a scheduling strategy (see Section 3.4) 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 the policy of a game-agent (i.e., a player) through GP. For the single-player game, the test cases constitute the game levels whereas for the two-player game, they are the opponent player policies. 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

[Dabas et al., 2022]. 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 policies is RL [Ghory, 2004; Thill et al., 2012], although various other strategies have been adopted over years [Schneider and Rosa, 2002; Sommerlund, 1996; Stenmark, 2005]. However, to the best of our knowledge, GP has never been used to evolve policies for this task, except for a preliminary study [Abdelsadek, n.d.].

Connect-4 is a two-player game on a 7×6 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 4.5×10^{12} possible board configurations [Edelkamp and Kissmann, 2008]. While other games, like chess or Go, provide a larger search space [Allis, 1994], we selected Connect-4 as a reasonable middle ground between the complexity of the task and ease of analysis. For this study, we use a simplified 6×6 board to reduce the computational burden of the experiments.

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 [Togelius et al., 2009] along with several competitions based upon it [Karakovskiy and Togelius, 2012]. This ignited researchers' interest in developing agents to solve the problem environment, yielding various contributions over the years [Togelius et al., 2010, 2013]. Among them, two contributions have encompassed GP: Perez et al. [2011] relied on grammatical evolution for optimizing policies in the form of behavior trees, whereas de Freitas et al. [2018] utilized grammar-based GP to evolve the policy.

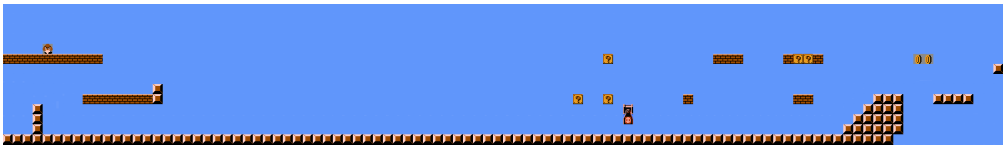
The game-play of Super Mario Bros involves navigating an agent—Mario—through different *levels* (see Figure 2), overcoming obstacles such as blocks and pipes, and defeating or avoiding enemies. We refer to each level as a *test case* in the following sections. A test case is considered solved if the agent is able to fully traverse it from left to right. To this end, the agent can perform zero or more actions at every game timestep among the following five (i.e., the action space has 2^5 elements): (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. [2009] for our evaluation.

3.2 Policy Search with GP

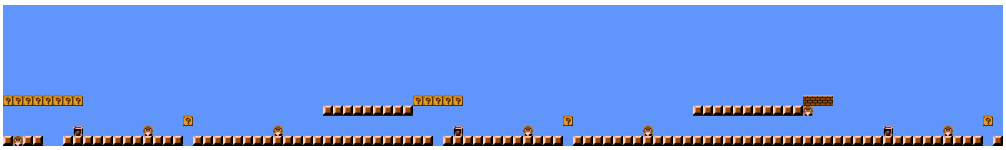
For both considered games, we evolve a game-agent policy with GP. We represent policies as linear programs or graphs, hence using the LGP and CGP as optimizers.

We call a policy the algorithm governing an agent immersed in an environment—which could possibly include an opponent, as in the Connect-4 case. We assume that the agent-environment pair evolves in discrete time: at each timestep, the agent takes an observation from the environment and produces an action, which is then applied to the environment. In this work, we consider numerical observation and action spaces: the policy is hence in general a function from $\mathbb{R}^{n_{\text{in}}}$ to $\mathbb{R}^{n_{\text{out}}}$, where n_{in} and n_{out} are respectively the dimensions of the observation and action spaces, i.e., the number of inputs and outputs of the policy.

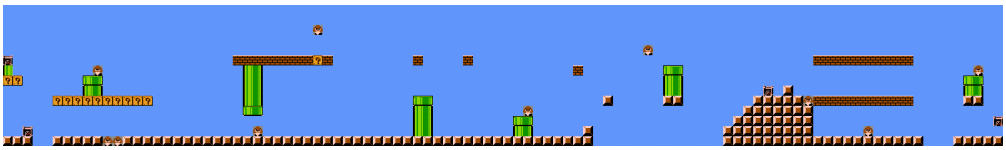
We express policies with graphs obtained with CGP or linear programs obtained with LGP. In both cases, we encode the policies through int strings, i.e., we search a genotype space $G \subset \mathbb{N}^p$ and we map each genotype $\boldsymbol{\theta}$ to a policy π using the standard LGP or CGP genotype-phenotype mapping function (see Sections 2.1.1 and 2.1.2).



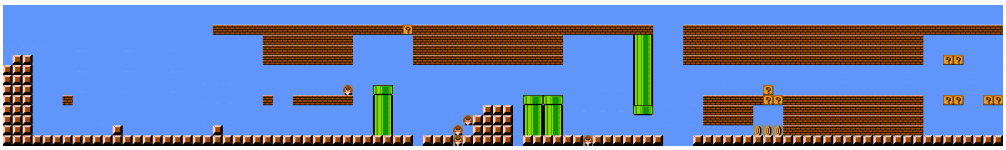
(a) Trivial: “little blocks, no enemies, no pipes”.



(b) Easy: “little blocks, some enemies, no pipes”.



(c) Medium: “little blocks, some enemies, some pipes”.



(d) Difficult: “many blocks, many enemies, many pipes”.

Fig. 2. Sample of LLM-generated test cases for Super Mario Bros for each difficulty, with corresponding prompts.

For evolving the policy, we use a mutation-only genetic algorithm (GA) with elitism, represented in Algorithm 1. After having initialized the population of n_{pop} individuals, we iteratively evolve it for n_{gen} generations. At each generation, we build an offspring of $(1 - \rho_{\text{elite}})n_{\text{pop}}$ individuals mutating parents chosen with tournament selection of size n_{tour}. Finally, we build the population for the next generation by merging the offspring and the $\rho_{\text{elite}}n_{\text{pop}}$ elite individuals of the current population. When assessing the fitness of an individual, we use a fitness function f which operates on the policy π and one or more test cases taken from the curriculum. We provide the details on how we instantiate this **evolutionary algorithm (EA)** with specific parameters values in Section 4.1. We describe the specific fitness function used for the two games considered in this study in the following sections.

Connect-4. For Connect-4, we evolve the policy to learn the optimal column for placing a piece on the board. We make the evolving policy compete against one or more predefined opponent policies, aiming for it to outplay them. At each timestep of the game, we query the policy 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 agent, and -1 a piece placed by the opponent. Regarding the agent’s *actions*, we consider the outputs from the final $n_{\text{out}} = 36$ registers (the size of the board) for LGP or from the $n_{\text{out}} = 36$ output nodes for CGP. For determining the action, i.e., the board column where to put the piece, we take the index of the largest value among outputs modulo the number of columns (i.e., 6): intuitively, this corresponds to choosing the “hottest column” for

Algorithm 1: The mutation-only GA used in this work which takes a fitness function f and returns a policy $\pi^\star$. The EA uses as parameters the population size n_{pop}, the number n_{gen} of generations, the relative size ρ_{elite} of the elite, the tournament size n_{tour}, and the curriculum $(\langle T_i, b_i \rangle)_i$ of cases. The functions INIT() and MUTATE depends on the genotype space and the function MAP() represents the representation. The function CASE() picks one or more test cases from the curriculum.

function EVOLVE($f; n_{\text{pop}}, n_{\text{gen}}, \rho_{\text{elite}}, n_{\text{tour}}, (\langle T_i, b_i \rangle)_i$)
  $P \leftarrow$ INIT(n_{pop}) ▷ initialize population
  for n_{gen} times **do** ▷ iteratively evolve the population
    $P_{\text{elite}} \leftarrow$ BESTS($P, \rho_{\text{elite}} n_{\text{pop}}$)
    $P_{\text{offspring}} \leftarrow \emptyset$ ▷ build offspring
    for $(1 - \rho_{\text{elite}}) n_{\text{pop}}$ times **do**
      $(\boldsymbol{\theta}, \pi, q) \leftarrow$ TOURNAMENT(P, n_{tour}) ▷ select parent
      $\boldsymbol{\theta}' \leftarrow$ MUTATE($\boldsymbol{\theta}$)
      $\pi' \leftarrow$ MAP($\boldsymbol{\theta}'$)
      $q' \leftarrow f(\pi'; \text{CASE}((\langle T_i, b_i \rangle)_i))$ ▷ compute fitness of child on a case
      $P_{\text{offspring}} \leftarrow P_{\text{offspring}} \cup \{(\boldsymbol{\theta}', \pi', q')\}$
    end for
    $P \leftarrow P_{\text{elite}} \cup P_{\text{offspring}}$
  end for
  $(\boldsymbol{\theta}^\star, \pi^\star, q^\star) \leftarrow$ BESTS($P, 1$)
  return $\pi^\star$ ▷ return the best policy
end function

the policy. To measure the quality of a policy, 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_i, where n_{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 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 policy of an agent to navigate test cases. We query the policy at every game timestep, 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 8×8 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 LGP or the n_{out} output nodes for CGP. For each of these outputs, 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 a policy, 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 a level.

3.3.1 Connect-4. To generate test cases, i.e., opponent policies, for the Connect-4 agent we use ChatGPT-4 [Achiam et al., 2023], Llama-3.1-Instruct-8B, and Llama-3.1-Instruct-405B [Dubey et al., 2024]. These models share some fundamental similarities but differ significantly in their accessibility, size, and operational characteristics. All three models are based on transformer architectures and are instruction-tuned, optimizing them for following specific prompts and generating appropriate responses. ChatGPT-4 is a proprietary model developed by OpenAI with limited access through API or subscription services. It requires an Internet connection and operates on a pay-per-use model. The exact size and architecture of ChatGPT-4 are not publicly disclosed, though it is believed to be significantly larger than its predecessors. In contrast, the Llama-3.1-Instruct models (developed by Meta) are open source models, allowing for much more flexible use and customization since users have access to all aspects of the model. These models can also be downloaded for free and run locally, so no Internet connection is required. Llama-3.1-Instruct-405B, with $\approx 405 \times 10^9$ parameters and a context window of 128,000 tokens, is designed for complex and resource-intensive tasks, whereas Llama-3.1-Instruct-8B, with $\approx 8 \times 10^9$ parameters, offers a lightweight alternative for less demanding applications with lower computational requirements.

Before proceeding further into this 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 investigate whether an LLM like ChatGPT-4 and Llama-3.1-Instruct can effectively generate opponents that enable a GP agent to learn and aim to compare and contrast the opponents generated by the different LLMs. To generate effective opponents of different difficulties, we began by communicating to the LLMs our requirement for diverse opponent strategies in Connect-4. We instructed each LLM to generate opponents with progressively increasing complexity, categorized as follows: (1) an *easy* (simple) opponent; (2) a *medium* opponent, building upon the easy opponent; and (3) a *difficult* opponent, improving the medium opponent.

For the Llama-3.1-Instruct-405B model, we additionally requested an *expert* opponent, designed to be more challenging than the difficult opponent. We chose to test this additive approach with the 405B model rather than the 8B model, as the greater complexity of 405B likely enables it to handle more sophisticated requests. Conversely, we did not request an expert opponent from ChatGPT-4, as there is no guarantee that the current version of ChatGPT-4 matches the version used to generate the initial three opponents. The specifics of the prompting strategy are outlined and discussed in the Appendix. We note that the LLM also provided the opponent policy implementation (in a specific programming language, provided as input by the user). We then embedded the provided code within our framework with minor post-processing needed to align them with the game simulation interface. We remark that we did not ask the LLMs to produce opponent policies in the form of LGP programs or CGP graphs.

To evaluate whether the generated opponent policies align with the intended difficulty levels, we conducted a preliminary analysis involving several matchups: against a random player—one that selects moves entirely at random, choosing a column for placing the piece without any strategic consideration; against themselves; and against each other. We report the results in Figures 3–5.

More specifically, Figure 3 reports the percentage of games won by policies of different difficulty levels, generated by various LLMs, against the random player. The 8B policies exhibit only a slight progression in performance as difficulty increases, suggesting potential inconsistencies in the expected difficulty progression for this model. In contrast, the 405B policies win nearly every game

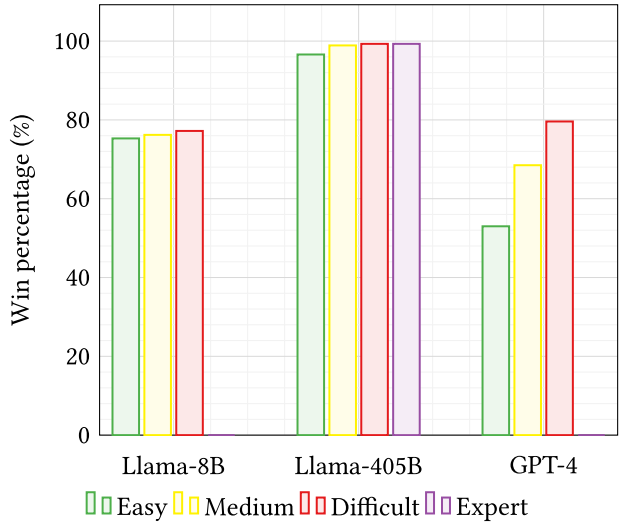


Fig. 3. Percentage of games won by policies generated by Llama-3.1-Instruct-8B (Llama-8B, in short), Llama-3.1-Instruct-405B (Llama-405B, in short), and ChatGPT-4 (GPT-4, in short), against the random player.

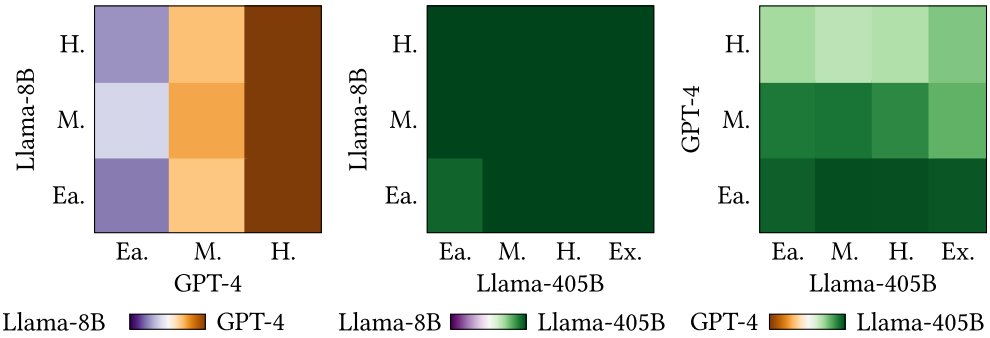


Fig. 4. Win ratio for games played between policies of various difficulties generated by different LLMs. Each heatmap corresponds to one combination of LLMs, reported as the x- and y-labels. The more intense either color (indicated in the legend below each map), the higher the win ratio for the policy generated by the corresponding LLM; the more white the tile the more balanced is the comparison (with ties or with an equally distributed win ratio among the considered policies).

against the random player. This highlights the overall strength of these policies but also indicates that even the easiest 405B policies are highly challenging. Meanwhile, the ChatGPT-4 policies demonstrate a clear and consistent progression in the number of wins as the generated policy difficulty increases. However, they never reached the same performance level as the 405B policies, suggesting that ChatGPT-4 policies might be overall simpler and less robust.

Figure 4 provides additional insight by presenting a heatmap of win ratios for games played between policies of varying difficulty levels, generated by different LLMs. The left plot shows that ChatGPT-4 policies exhibit a clear progression, with higher-difficulty policies consistently achieving more wins against 8B policies. Notably, the GPT-4 difficult policy wins every single game in this comparison. The middle plot compares policies generated by the 8B and 405B models, revealing that the 405B policies dominate their 8B counterparts across all difficulty levels. These

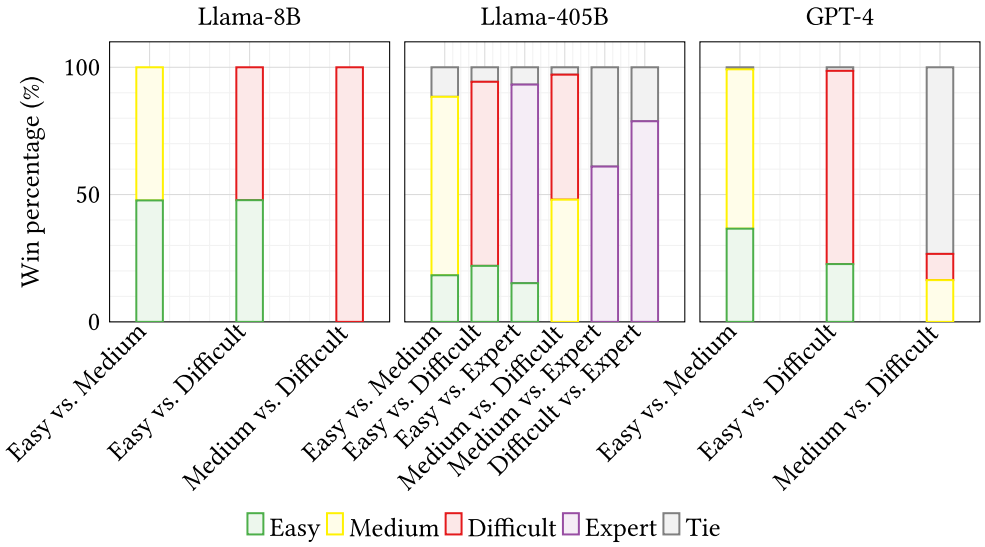


Fig. 5. Win ratio for games played between the easy, medium, difficult (and expert—only for Llama-405B) policies generated by each of the considered LLMs.

results show the significant improvements in policy generation capabilities associated with the larger model. The right plot highlights that 405B policies consistently outperform the ChatGPT-4 policies by a large margin. However, the win rate of the 405B policies decreases as the difficulty of the ChatGPT-4 agent increases, indicating that the harder GPT-4 policies offer greater resistance.

Figure 5 shows the win ratios for games played between the easy, medium, difficult, and (for 405B) expert policies generated by each of the considered LLMs. For the 8B model, the results reveal limited differentiation in policy performance across difficulty levels: the easy policy performs as well as the medium and difficult policies, while the difficult policy consistently defeats the medium one. In contrast, the Llama-405B model exhibits a clearer progression in difficulty: higher-difficulty policies typically win more games in each pairwise matchup, with the exception of the medium vs. difficult comparison, where both policies perform equally well. For ChatGPT-4, the easy policy is consistently defeated by both the medium and difficult policies. In medium vs. difficult games, most result in ties, with the medium policy wins slightly more of the remaining games.

These results demonstrate the complexities involved in generating game-playing policies with distinct difficulty levels using language models. While ChatGPT-4 policies align more consistently with the expected difficulty progression, they are overall less challenging than the 405B policies. In contrast, the 405B policies exhibit greater difficulty but fail to differentiate sufficiently between the medium and difficult levels. The inconsistent performance of the 8B policies further highlights the importance of model size in generating effective and well-defined difficulty levels.

3.3.2 Super Mario Bros. For generating the test cases for Super Mario Bros we rely on *MarioGPT* [Sudhakaran et al., 2024], 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,” and “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 [Sudhakaran et al., 2024], 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. [2024], 88% of the generated test cases are playable, i.e., there exists a path that allows its traversal. Before using a generated level within a GP optimization we test its feasibility, as suggested by the authors of MarioGPT, by running Robin Baumgarten's A* agent [Togelius et al., 2009] on it. If it is not solvable, we repeat the generation, decreasing the MarioGPT temperature parameter 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 2. 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 levels 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 [Togelius et al., 2009]. 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 in a curriculum to achieve successful policy search with GP. We explored several ways to assemble test cases in curricula, which we describe below. We remark that we did not rely on LLMs for building curricula.

Connect-4. For Connect-4, we have three LLM-generated opponents of increasing difficulty for ChatGPT-4 and Llama-3.1-Instruct-8B, and four for Llama-3.1-Instruct-405B. Thus, a natural approach is to organize them in a curriculum, $(\langle T_i, b_i \rangle)_i$, with $i = 1, 2, 3$ (and 4, for Llama-3.1-Instruct-405B), where $T_1 = \{t_e\}$ includes only the easy opponent t_e, $T_2 = \{t_m\}$ includes only the medium opponent t_m, $T_3 = \{t_d\}$ includes only the difficult opponent t_d, and $T_4 = \{t_{\text{exp}}\}$ includes only the expert opponent t_{exp}.

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_{gen} of generations for the evolution of the GP population and distributing these generations across different opponents, with each b_i indicating the percentage of generations consumed on the ith opponent. For brevity,

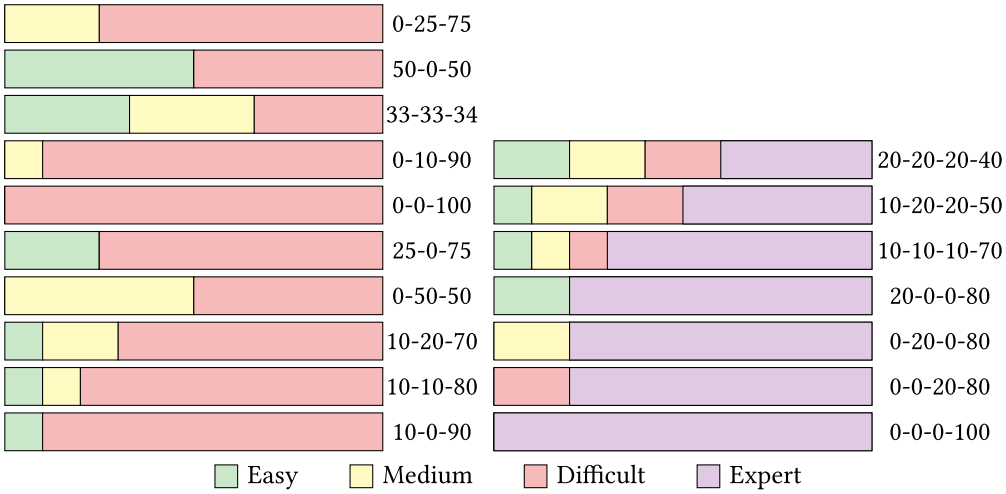


Fig. 6. Graphical representations of the considered curricula for Connect-4. The color represents the difficulty of the opponent, the size of the rectangle represents its duration in the schedule. The left side of the figure shows curriculum strategies involving three opponents (b_1-b_2-b_3) whereas the right side shows strategies with four opponents (b_1-b_2-b_3-b_4).

we denote by b_1-b_2-b_3-b_4 a curriculum $(\langle\{t_e\}, b_1\rangle, \langle\{t_m\}, b_2\rangle, \langle\{t_d\}, b_3\rangle, \langle\{t_{\text{exp}}\}, b_4\rangle)$—when $b_i = 0$, the corresponding ith pair is absent from the curriculum. For the LLMs with only three opponents, b_4 is always set to zero. For Llama-3.1-Instruct-405B, we analyze the curriculum effect both using only the first three opponents (to enable comparison with other LLMs) and in a separate experiment that includes the expert opponent in the curriculum. We also use GP for evolving policies tested solely against the difficult LLM-generated opponent, represented as 0-0-100 (or 0-0-0-100 for Llama-3.1-Instruct-405B with the expert opponent), to evaluate the impact of the curriculum approach. We show in Figure 6 the curricula implemented for Connect-4.

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 by building curricula with levels of increasing difficulty. We define three curricula, together with two baselines consisting only of the last stage, i.e., of the most difficult one, T_4.

The first two curricula consists of four increasing difficulties $(\langle T_i, b_i\rangle)_i$, with $i = 1, \ldots, 4$. 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, which we set to 3) 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 (C-PAR), 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_{stag} generations (C-SEQ).

Concerning the third curriculum—which we call C-GRAD—it builds on C-PAR with the difference that the transitions between subsequent difficulties are made more gradual. Thus, it still consists of four main difficulties T_1, T_2, T_3, T_4, yet the change between difficulty level is made by changing *only one* test case at a time. In other words, we define intermediate levels of difficulty T_i^ρ, where ρ

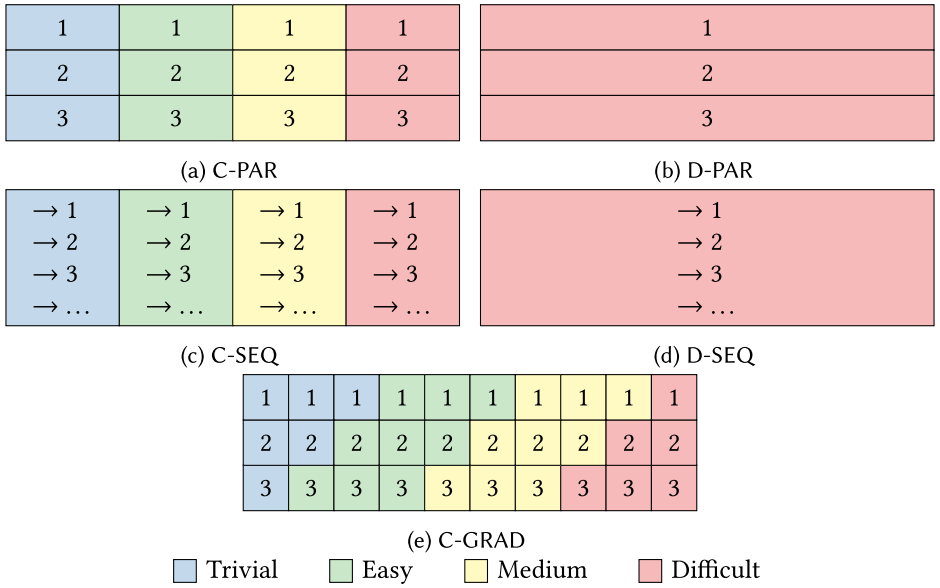


Fig. 7. Graphical representations of curricula for Super Mario Bros. The color represents the difficulty of the environment; each number n indicates the nth test case of a given difficulty. Stacked rectangles represent parallel evaluations, whereas we use arrows (→) to indicate sequential skips between test cases.

indicates the ratio of test cases of the next difficulty level, $i + 1$, replacing those of difficulty i. We consider $\rho \in \{0, 1/3, 2/3\}$ for $i \in \{1, 2, 3\}$ and $\rho = 0$ for $i = 4$, totaling $3 \cdot 3 + 1 = 10$ difficulty levels. As a consequence, the computational budget is divided accordingly, and less time is spent on each combination.

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 $i = 1, \ldots, 4$, $b_1 = \cdots = b_4 = 25$, and parallel test case evaluation at each T_i;
(2) *C-SEQ*, a CL schedule $(\langle T_i, b_i \rangle)_i$, with $i = 1, \ldots, 4$, $b_1 = \cdots = b_4 = 25$, and sequential test case evaluation at each T_i;
(3) *C-GRAD*, a CL schedule $(\langle T_j, b_j \rangle)_j$, with $T_j = T_i^{\rho}$, $i = 1, \ldots, 4$, $b_1 = \cdots = b_{10} = 10$, $\rho \in \{0, 1/3, 2/3\}$ for $i \in \{1, 2, 3\}$ and $\rho = 0$ for $i = 4$, and parallel test case evaluation at each T_j;
(4) *D-PAR*, the C-PAR version considering only T_4, i.e., $b_1 = \cdots = b_3 = 0$ and $b_4 = 100$; and
(5) *D-SEQ*, the C-SEQ version considering only T_4, i.e., $b_1 = \cdots = b_3 = 0$ and $b_4 = 100$.

For each of these, we equally divide the total computational budget among the stages of the curriculum. See Figure 7 for a visual overview of these curricula.

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 doing policy search with GP? If so, is the choice of the LLM affecting the final outcome?
RQ2 How can we best schedule the training process to leverage LLM-generated test cases effectively? Is CL a viable strategy?

RQ3 Are the results consistent across different GGP-representations?

We detail the experimental setup and procedure in the following sections.

4.1 Parameters and Procedure

For optimizing the policies, we rely on the mutation-only GA with elitism described in Section 3.2 (see Algorithm 1). We use a population of size $n_{\text{pop}} = 50$ for Connect-4 and $n_{\text{pop}} = 100$ for Super Mario Bros for n_{gen} generations. To determine n_{gen}, we fix the total amount of fitness evaluations n_{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 genotype in the population initialization phase, we sample a uniform distribution over the allowed integer values for each position. We use an elite of $0.1n_{\text{pop}}$, i.e., we set $\rho_{\text{elite}} = 0.1$ and a tournament size $n_{\text{tour}} = 3$. As mutation, we use int-flip mutation. For LGP, 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 ($p_i = 0.1$). For CGP, we assign a different mutation probability to the output nodes ($p_o = 0.3$), to the inputs of function nodes ($p_i = 0.1$), and to the functions ($p_f = 0.1$).

Concerning the representations, we employ the LGP and CGP described in Sections 2.1.2 and 2.1.1. We consider the following function set for both representations $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \sqrt{\bullet}^*, \bullet < \bullet, \bullet > \bullet, \bullet < 0, \bullet > 0\}$, 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. For both representations we add two constant inputs to each observation, namely $\{0.1, 1\}$. These constants are treated like the agent regular observations and can contribute to the policy computation. We set $n_{\text{reg}} = n_{\text{in}} + n_{\text{const}} + n_{\text{out}} + 5$ and $n_{\text{lines}} = 20$ for LGP, which results in a genotype size of $|\boldsymbol{\theta}| = (2 + m_{\text{ar}})n_{\text{lines}} = (2 + 2)20 = 80$. For CGP, we use a 1D grid representation with $n_{\text{nodes}} = 50$, yielding a genotype size of $|\boldsymbol{\theta}| = n_{\text{out}} + (1 + m_{\text{ar}})n_{\text{nodes}} = n_{\text{out}} + (1 + 2)50 = n_{\text{out}} + 150$. This corresponds to a total genotype size of $|\boldsymbol{\theta}| = 186$ for Connect-4 and $|\boldsymbol{\theta}| = 155$ for Super Mario Bros, based on the respective values of n_{out}.

Concerning the curricula, we consider the scenarios described in Section 3.4. For Connect-4, we experiment with 10 budget allocation combinations across stages for three opponents (i.e., in the form b_1-b_2-b_3) and seven combinations for four opponents (in the form b_1-b_2-b_3-b_4). For Super Mario Bros, we set n_t, i.e., the number of test cases per difficulty, to 3 as a reasonable tradeoff 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 three moves, a well-known approach that involves predicting and negating the opponent's moves for optimal play [Nasa et al., 2018]. For Super Mario Bros we consider eight test cases (two for each difficulty) not seen during evolution generated by MarioGPT along with 15 test cases extracted from the original Super Mario Bros game: we keep the same 23 game levels for assessing all the evolved policies.

To obtain statistically significant results, we perform 30 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]

[1] https://doi.org/10.5281/zenodo.17182432, https://doi.org/10.5281/zenodo.17141432.

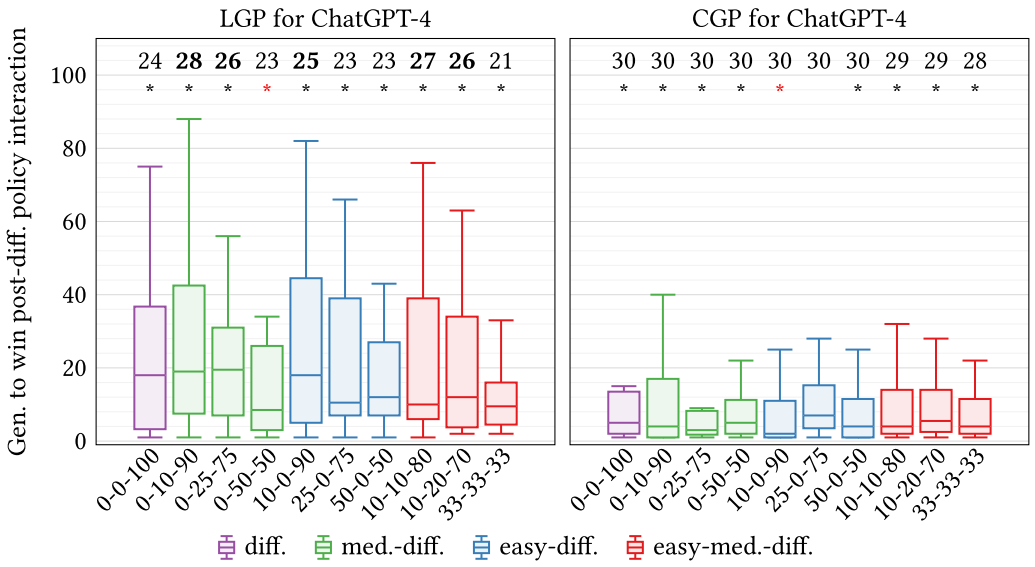


Fig. 8. Distribution of the generation for the first victory of the best GP agent against the difficult policy, across 30 runs. The results refer to opponents generated by ChatGPT-4. On the left, we report the results using LGP, on the right using CGP. Above each box, we show the number of runs which gave a policy able to win against the difficult opponent: we highlight in bold the cases for which this number is greater than the non-curriculum strategy (0-0-100). The variant with the fastest median convergence speed (lowest number of generations for the first victory) is marked with a red asterisk (*); variants with statistically equivalent convergence speed to the best one are marked with a black asterisk (*).

4.2 Results

4.2.1 Connect-4. To evaluate the effectiveness of opponents generated by the LLM, we first assess the evolved policies' ability to win a game. This analysis also aims to explore the impact of the specific LLM used to generate the opponents, as well as the GGP variants adopted for the GP agents. We present results from different LLM: ChatGPT-4 in Figure 8, Llama-3.1-Instruct-8B in Figure 9, and Llama-3.1-Instruct-405B in Figure 10. In each figure, we show on the left the results for policies evolved with LGP, on the right for the ones evolved with CGP.

Specifically, the plots report the earliest generation in which the best policy in the GP population wins against the difficult opponent across 30 runs, using various curricula. This includes a baseline where training occurs solely against the difficult opponent (0-0-100). Since curriculum-based training involves playing against the difficult opponent only after some rounds against the easy and/or medium opponents, we present the earliest generation in which the agent wins, starting the count from its first encounter with the difficult policy: the lower, the more efficient the policy search. This ensures a fair comparison across the different curricula. 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 is a coarser measure of the effectiveness of the policy search process.

While the number of victories is the primary criterion for identifying the best agents, we are also partially interested in the convergence speed—that is, how quickly an agent achieves its first win. This information is summarized in the boxplots, which display the distribution of the earliest winning generations. To assess the statistical significance of differences among variants (namely, among their number of generations for the first victory), we perform pairwise Mann-Whitney U

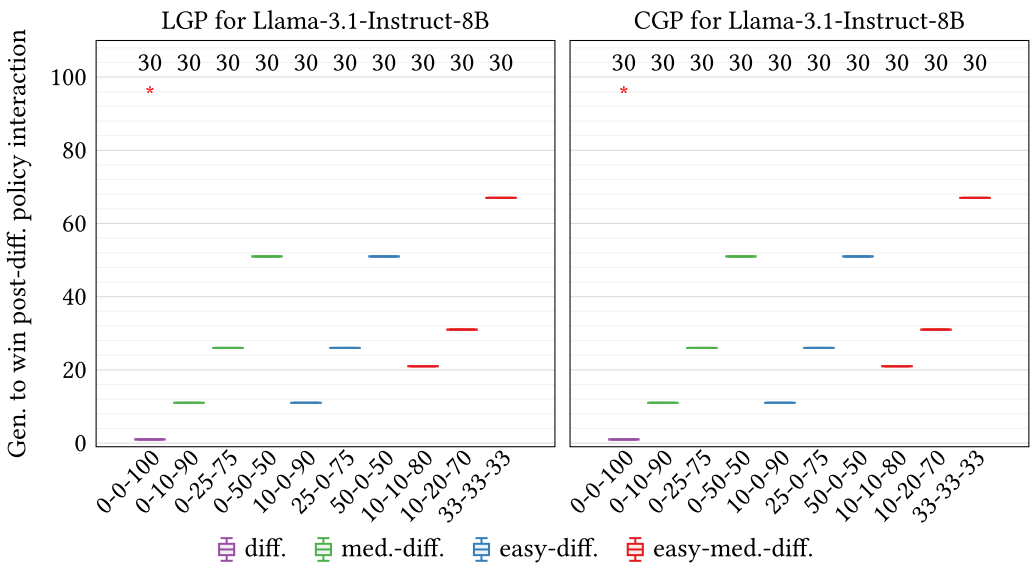


Fig. 9. Distribution of the generation for the first victory of the best GP agent against the difficult policy, across 30 runs. The results refer to opponents generated by Llama-3.1-Instruct-8B. On the left, we report the results using LGP, on the right using CGP. Above each box, we show the number of runs which gave a policy able to win against the difficult opponent: we highlight in bold the cases for which this number is greater than the non-curriculum strategy (0-0-100). The variant with the fastest median convergence speed (lowest number of generations for the first victory) is marked with a red asterisk (*); variants with statistically equivalent convergence speed to the best one are marked with a black asterisk (*).

tests with a significance level of $\alpha = 0.05$. In the boxplots, the variant with the best median is marked with a red asterisk (*), while a black asterisk (*) indicates variants whose performance distributions are statistically equal to the best variant. Variants without any asterisk show statistically slower convergence.

Comparison of LLMs. Regarding the results obtained with ChatGPT-4, Figure 8 confirms GP capability to evolve effectively from LLM-generated opponents across various curricula, including the case where a curriculum is not adopted, ensuring success against the difficult policy. Examining the performance of different GGP representations, we observe that CGP policies consistently outperform their LGP counterparts. Notably, the CGP representation enables agents to defeat the difficult opponent consistently, regardless of the use of CL. This victory is achieved early in the evolution, some generations after the first encounter with the difficult opponent. However, training against easier opponents remains advantageous within some curricula, as it reduces the time required to defeat the difficult policy compared to training exclusively against it. On the other hand, LGP agents do not achieve consistent victory across all curricula, allowing us to investigate the impact of CL strategies in more detail. Figure 8 shows that certain curricula improve the agent's learning efficiency and victory rate over difficult-only curriculum (0-0-100). Our findings suggest that an initial session with a weaker opponent, followed by progression to a more challenging one, is effective when kept brief. Excessively reducing training against the difficult opponent and spending too many fitness evaluations against weaker opponents is counterproductive, as agents do not engage enough with the difficult opponent to learn how to defeat it. Additionally, structuring the curriculum to include interactions with the easy opponent proves more beneficial compared

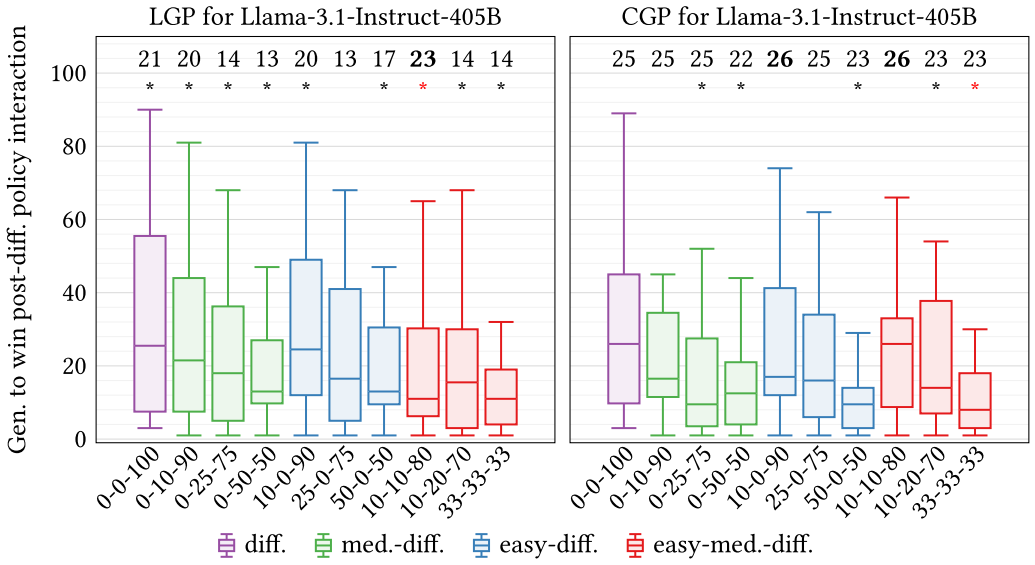


Fig. 10. Distribution of the generation for the first victory of the best GP agent against the difficult policy, across 30 runs. The results refer to opponents generated by Llama-3.1-Instruct-405B. On the left, we report the results using LGP, on the right using CGP. We use bold for the x-labels relative to the curricula differing in a statistically significant way from the non-curriculum strategy (0-0-100). The variant with the fastest median convergence speed (lowest number of generations for the first victory) is marked with a red asterisk (*).

to starting directly with a medium opponent. The plots further confirm that training with easier opponents before encountering the difficult one allows agents to defeat the difficult opponent earlier than the baseline (upon their first encounter).

Focusing on the Llama-3.1-Instruct-8B model (Figure 9), the behavior differs significantly from previous results. In this case, policies evolved by GP—regardless of their representation or the curricula used—consistently defeat the difficult opponent immediately after their first encounter with it. This suggests that the LLM failed to generate truly challenging difficult opponents, aligning with the findings of the preliminary analysis presented in Section 3. In fact, Figure 4 shows that the difficult 8B opponent is consistently defeated by the difficult GPT-4 opponent and often also by the medium GPT-4 opponent. This indicates that not only do LLMs generate different test cases when prompted with a specific difficulty level, but also these test cases themselves vary intrinsically in their level of challenge.

Finally, for the Llama-3.1-Instruct-405B model, we recall that the LLM was prompted to generate also a more challenging opponent, referred to as the expert opponent. Therefore, we present two separate analyses: the first focuses only on the easy, medium, and difficult opponents (Figure 10), to allow for a direct comparison in the generation of opponents across different LLM, while the second includes the expert opponent (Figure 11) to investigate how CL perform when additional opponents are used. We recall that, since the expert opponent is more challenging to defeat, the experiments in Figure 11 were conducted with a population of 100 individuals, compared to 50 used in the other experiments. As in Figure 8, a key observation emerging from both Figures 10 and 11 is that the GP representation plays a crucial role on agents performance. The best results are achieved by CGP, which consistently outperforms its LGP counterpart. These plots also show that the GP representation influences the effectiveness of CL in defeating the expert opponent. Interestingly,

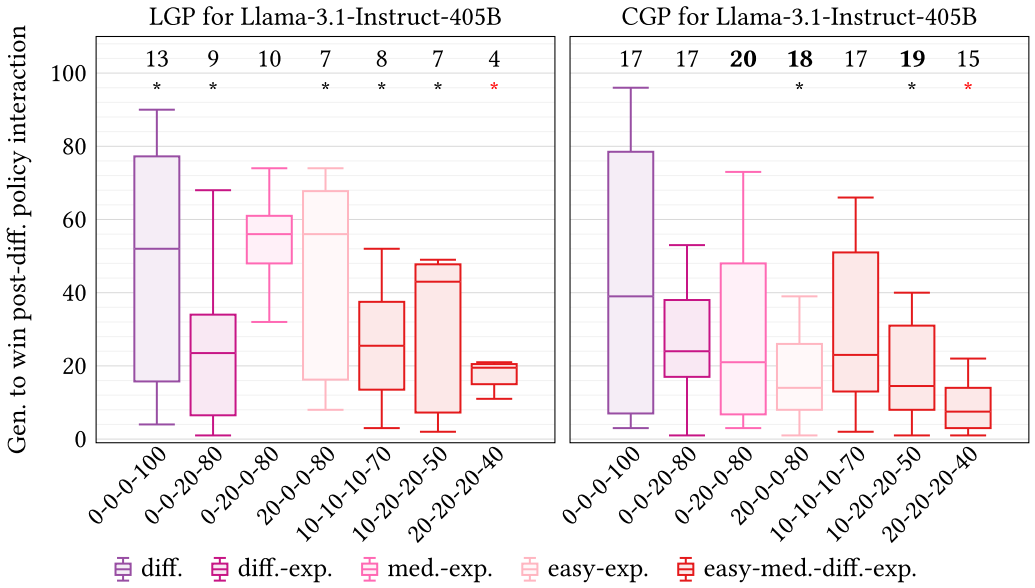


Fig. 11. Distribution of the generation for the first victory of the best GP agent against the expert policy, across 30 runs. The results refer to opponents generated by Llama-3.1-Instruct-405B. On the left, we report the results using LGP, on the right using CGP. Above each box, we show the number of runs which gave a policy able to win against the difficult opponent: we highlight in bold the cases for which this number is greater than the non-curriculum strategy (0-0-100). The variant with the fastest median convergence speed (lowest number of generations for the first victory) is marked with a red asterisk (*); variants with statistically equivalent convergence speed to the best one are marked with a black asterisk (*).

with LGP, policies evolved exclusively against the expert opponent (baseline) are more successful in defeating it by the end of evolution. In contrast, with CGP, CL proves to be beneficial, leading to better performance when it is used compared to when it is not.

Overall, GP agents evolved playing against GPT-4 opponents achieve a higher win rate against its difficult opponent (Figure 8) compared to those trained against 405B (Figure 10). This difference can be attributed to the different intrinsic difficulty of the opponents generated by the two LLMs. As shown in Figure 4, opponents generated by 405B tend to be more challenging than those produced by GPT-4. Additionally, Figure 3 reveals that all opponents (easy, medium, difficult, and expert) generated by 405B exhibit similar and relatively high win percentages against the random player. In contrast, GPT-4 shows a clear progression in difficulty, with increasing difficulty levels corresponding to progressively higher win rates against the random player. This discrepancy also explains the reduced effectiveness of CL in 405B compared to GPT-4. While 405B can generate opponents of increasing difficulty (Figure 3), even its easiest opponents perform exceptionally well against the random player. This limits the impact of CL, as agents lack the opportunity to begin training against truly easy opponents.

Curricula Comparison. To investigate the learning progress of different curricula, we report in Figure 12 the progression of the fitness of the best GP agent, for one random seed. We decided to report results on an individual seed rather than aggregate values, as learning behavior shows significant variation across seeds, and analyzing an individual case is more informative. The analysis includes results for ChatGPT-4 and Llama-3.1-Instruct-405B models and for both GGP representations. Results for the Llama-3.1-Instruct-8B model are omitted, as Figure 9 demonstrates

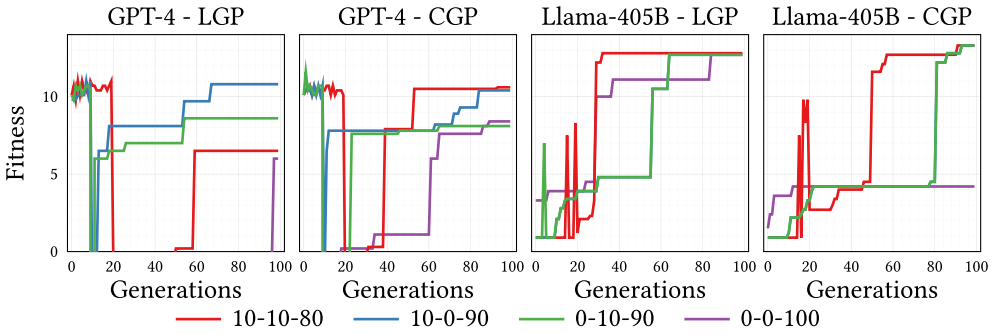


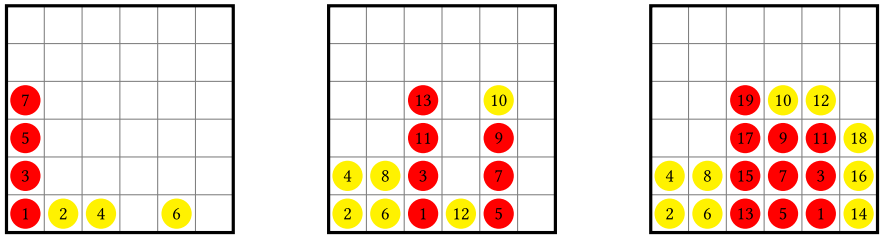
Fig. 12. Fitness across generations for the best policy in a fixed run, using for ChatGPT-4 and Llama-3.1-Instruct-405B each with both GGP representations.

that agents consistently win against difficult opponents upon their first encounter, making fitness evolution uninformative in this context. For each category, we present the fitness progression of the most successful curriculum and, as a baseline, include the performance of agents trained exclusively against the difficult opponent. Interestingly, the most efficient curricula for a given category are consistent across both GGP representations and LLM models used for opponent generation. Specifically, for the easy-diff. category, the 10-0-90 curriculum is the most effective; for the med.-diff. category, 0-10-90 performs best; and for the easy-med.-diff. category, 10-10-80 is the most successful. For all the experiments, these curricula result in an equal or (more often) higher proportion of winning agents compared to the baseline, indicating that it is beneficial to include preliminary training against easier opponents, but only if it remains short.

Most seeds result in winning agents (except for the baseline in 405B with CGP), but the fitness scores differ due to our definition of fitness, which rewards wins and penalizes for opponent alignments. The baseline agent consistently exhibits equal or lower fitness compared to agents trained with a CL strategy, suggesting that the baseline is less effective in defending against opponent strategies. Regarding the results obtained with GPT-4, 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. Upon transitioning to the difficult opponent, there is an initial drop in fitness; however, after a few generations, these agents almost always adapt and return to winning, demonstrating rapid improvement to the increased challenge. In contrast, the 405B results reveal that agents are not always able to defeat the easiest opponents. This aligns with previous findings regarding the intrinsic difficulty of 405B opponents, where even the easy ones present a significant challenge.

Finally, in Figure 13 we report a game played by a winning best ChatGPT-4 LGP agent (with curriculum 10-0-90), tested against the three ChatGPT-4 policies used in this study. This game is reported exclusively for this LLM and GP representation, as the results for other configurations were very similar and do not provide additional insights. The goal here is simply to illustrate an example of the policy in action.

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 13 also offers a visual representation of the increasing difficulty of different LLM-generated opponents. The first game board shows a relatively straightforward win for our agent, reflecting an opponent designed to teach basic moves. The medium opponent is more challenging, yet the agent wins as it is not consistently blocked. The final game, against the difficult opponent, is the most interesting one as



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

Fig. 13. Best LGP 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.

it demonstrates the agent's ability to develop a strategic approach by creating multiple paths to victory.

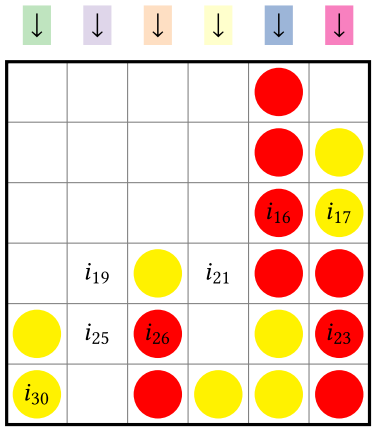
We also test our method against the minimax strategy, known for its optimal play. In 10 games each, both the baseline (0-0-100) and best curriculum (10-0-90) agent win 3 times. The results are not surprising given that minimax is an advanced strategy compared to the ones used against the agent during 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.

Example of an LGP Policy. In Figure 14, we analyze the behavior of an LGP policy evolved against the ChatGPT-4 difficult opponent using curriculum 10-0-90. This visualization aims to provide insight into the decision-making process of the evolved policy and how algebraic expressions support successful play. We report a board state—with input features ordered from top-left to bottom-right—together with the symbolic policy evolved by the agent. Each instruction can contribute to one or more output expressions, which represent scalar scores used to compute the probability of selecting each column. Specifically, each output is assigned to a column based on the register index modulo 6 (i.e., a register at index j contributes to column j mod 6). Each expression is highlighted in the same color as the arrow above the corresponding column, visually linking it to the action it influences.

The expressions reflect the behavior learned by the agent. For instance, column 2 is favored, receiving a constant term ($o_{31} = 1$) and a logarithmic term ($o_{25} = \log(|i_{21}|)$), which is always positive when the input i_{21} (a central cell) is occupied—either by the agent or the opponent. This reflects a preference for central columns, which offer more opportunities to build winning alignments. The expression $o_{26} = 1$ if $i_{16} < 0$ else 0 activates when the opponent occupies a specific side cell, prompting the agent to block potential threats while contesting a favorable position. In column 5, the agent combines a stacking heuristic (i_{25}), rewarding positions where it has already placed tokens, with i_{30}, which checks whether the bottom-left cell is occupied—often by the opponent, since the agent rarely plays in column 1. Overall, the policy integrates progress, defense, and board awareness using simple yet effective symbolic patterns.

4.2.2 Super Mario Bros. For Super Mario Bros, we did not consider multiple LLMs, given that to generate a well-formed level an off-the-shelf LLM would need to be fined tuned. Thus we disregard the first research question for this scenario. Concerning the second and the third research questions, we report various results in Figures 15 and 17, and Tables 1 and 2.

First, to evaluate whether the LLM-generated test cases are effective for evolving policies with GP in Super Mario Bros, we examine the top two plots of Figure 15, where we show the progression



(a) Connect 4 game.

```
def controller(inputs, r):
    r[0:38] = inputs
    r[72] = cos(r[17])
    r[74] = cos(r[64])
    r[69] = r[74] < r[1]
    r[48] = r[26] > r[23]
    r[51] = sqrt(r[72])
    r[68] = log(r[21])
    r[71] = sqrt(r[30])
    r[73] = r[44] > r[52]
    r[78] = sin(r[64])
    r[77] = r[25] + r[69]
    r[54] = r[61] * r[45]
    r[43] = r[60] > r[47]
    r[76] = r[23] < r[19]
    r[69] = r[16] < r[57]
    return r[-36:]
```

$o_5 = 1$ if $i_{26} > i_{23}$ else 0

$o_8 = \sqrt{\cos(i_{17})}$

$o_{25} = \log(|i_{21}|)$

$o_{26} = 1$ if $i_{16} < 0$ else 0

$o_{28} = |i_{30}|$

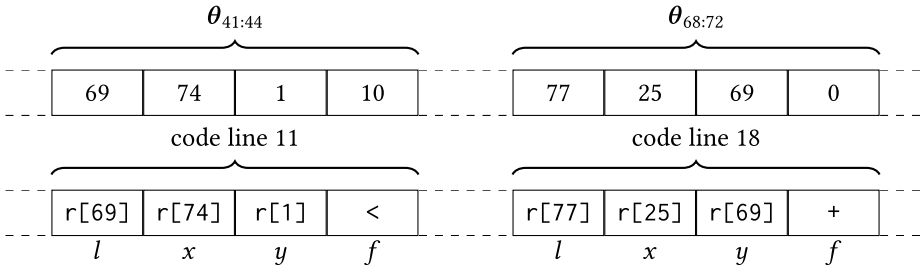
$o_{29} = \cos(i_{17})$

$o_{31} = 1$

$o_{33} = 1$ if $i_{23} < i_{19}$ else 0

$o_{34} = i_{25} + (1$ if $i_1 > 1$ else $0)$

(b) LGP policy.



(c) Portion of the genotype $\boldsymbol{\theta}$ with a sketch of the encoding process.

Fig. 14. Representation of a sample Connect 4 game with the inputs to the policy highlighted (a), a sample policy found by LGP shown in its original program form and as formulae (b), and a portion of the genotype encoding it (c). The test board state in (a) is encoded in row-major order from top-left to bottom-right, using +1 for the tokens of the agents (red), −1 for the ones of the opponent (yellow), and 0 for empty cells. Each formula of the LGP policy in (b) is highlighted using the same color as the arrow above the corresponding column in the board, indicating which expressions contribute to the score for that column (with column index determined by instruction index modulo 6). Finally, the portion of the genotype $\boldsymbol{\theta}$ relevant to one of the outputs (o_{34}) is shown in (c), along with the corresponding equation derivation process: from top to bottom, the raw genotype $\boldsymbol{\theta}$, its inline interpretation, a subsequence of instructions, the formula for the output o_{34}.

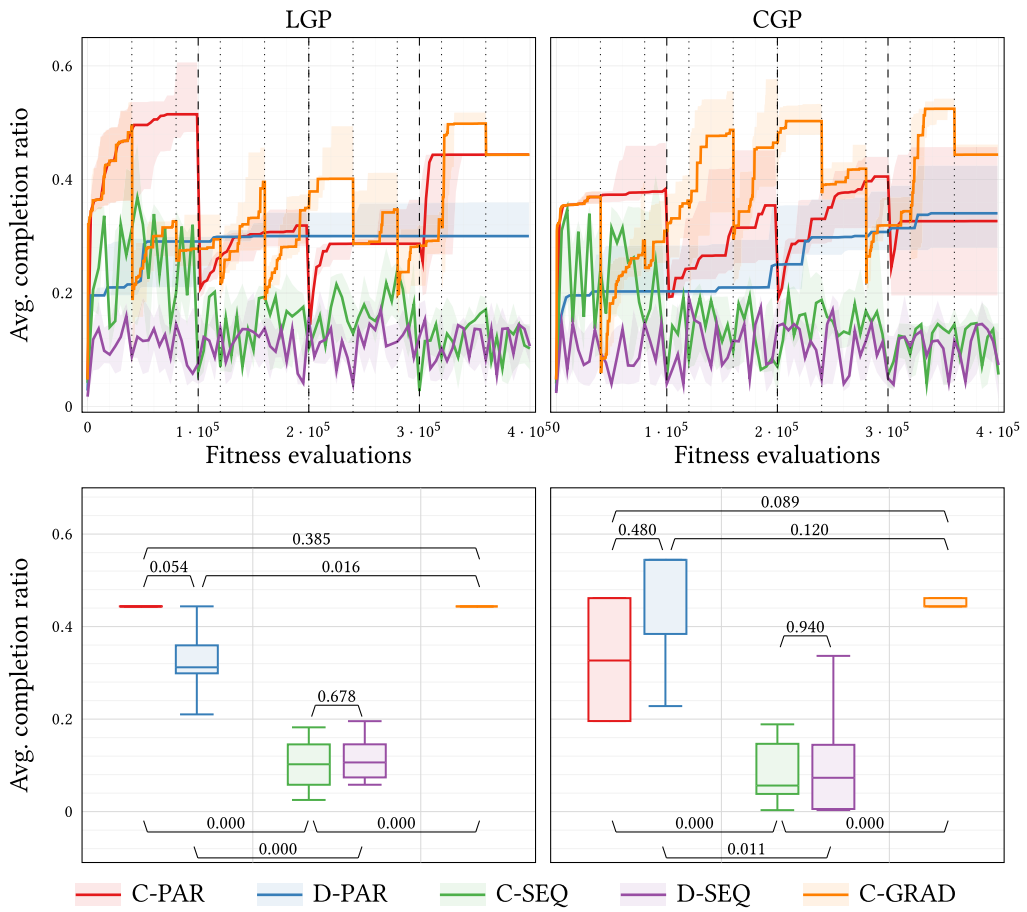


Fig. 15. Progression and final distributions of the average completion fraction (averaged across $n_t = 3$ test cases) achieved collected by the best GP policy in the population at each iteration; median and inter-quartile range across 10 independent runs. For C-PAR and C-SEQ (and C-GRAD) the vertical dashed (dotted) lines indicate the change of difficulty in the curriculum; 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.

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 C-PAR, D-PAR, and C-GRAD, 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, C-SEQ, and C-GRAD.

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, and for the gradual scheduling C-GRAD, as indicated by an increase in fitness over time for both GGP representations. Concerning the sequential scheduling cases, instead, the lines for both C-SEQ and D-SEQ appear noisy and unstable,

Table 1. Results of Statistical Tests (Pairwise Wilcoxon Rank-Sum Tests with Equality as Null Hypothesis) for the Data Shown in Figure 17

		LGP					CGP				
		C-PAR	C-SEQ	C-GRAD	D-PAR	D-SEQ	C-PAR	C-SEQ	C-GRAD	D-PAR	D-SEQ
Trivial	C-PAR	—	**≈0**	0.268	0.351	**≈0**	—	**0.002**	0.124	**0.041**	**≈0**
	C-SEQ	≈0	—	**0.003**	**0.003**	0.832	0.002	—	**≈0**	0.178	0.594
	C-GRAD	0.268	0.003	—	0.773	**0.002**	0.124	≈0	—	**≈0**	**≈0**
	D-PAR	0.351	0.003	0.773	—	**0.002**	0.041	0.178	≈0	—	0.066
	D-SEQ	≈0	0.832	0.002	0.002	—	≈0	0.594	≈0	0.066	—
Easy	C-PAR	—	0.155	0.177	0.402	**0.018**	—	**0.002**	0.166	0.415	**0.004**
	C-SEQ	0.155	—	**0.007**	0.520	0.389	0.002	—	**≈0**	**0.012**	0.761
	C-GRAD	0.177	0.007	—	**0.038**	**≈0**	0.166	≈0	—	**0.036**	**≈0**
	D-PAR	0.402	0.520	0.038	—	0.135	0.415	0.012	0.036	—	**0.021**
	D-SEQ	0.018	0.389	≈0	0.135	—	0.004	0.761	≈0	0.021	—
Medium	C-PAR	—	**≈0**	0.905	**≈0**	**≈0**	—	**≈0**	0.084	0.051	**≈0**
	C-SEQ	≈0	—	**≈0**	0.060	0.496	≈0	—	**≈0**	**0.011**	0.074
	C-GRAD	0.905	≈0	—	**0.002**	**≈0**	0.084	≈0	—	**≈0**	**≈0**
	D-PAR	≈0	0.060	0.002	—	0.184	0.051	0.011	≈0	—	**≈0**
	D-SEQ	≈0	0.496	≈0	0.184	—	≈0	0.074	≈0	≈0	—
Difficult	C-PAR	—	**0.020**	0.928	0.145	0.160	—	**0.024**	0.891	0.351	**≈0**
	C-SEQ	0.020	—	**0.015**	0.444	0.226	0.024	—	**0.028**	0.133	0.083
	C-GRAD	0.928	0.015	—	0.100	0.156	0.891	0.028	—	0.315	**≈0**
	D-PAR	0.145	0.444	0.100	—	0.779	0.351	0.133	0.315	—	**0.001**
	D-SEQ	0.160	0.226	0.156	0.779	—	≈0	0.083	≈0	0.001	—

Statistically significant differences are highlighted in bold.

Table 2. Ranking of Each Curriculum on 15 Original Super Mario Bros Levels

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Sum
LGP	C-PAR	2	**1**	2	**1**	**1**	3	**1**	**1**	**1**	3	3	**1**	3	**1**	**1**	27
	C-SEQ	**1**	5	4	5	3	4	4	5	**1**	4	4	4	5	5	4	59
	C-GRAD	4	3	**1**	**1**	5	**1**	**1**	3	4	**1**	2	**1**	2	2	4	37
	D-PAR	3	2	3	3	3	2	4	2	3	2	**1**	4	**1**	3	2	39
	D-SEQ	5	4	5	3	2	5	4	4	4	5	5	4	4	3	4	62
CGP	C-PAR	3	3	2	2	2	2	2	2	4	2	2	2	**1**	3	4	37
	C-SEQ	2	2	3	5	**1**	3	4	5	2	3	4	4	4	2	2	46
	C-GRAD	**1**	**1**	**1**	**1**	5	**1**	**1**	3	**1**	**1**	**1**	**1**	3	**1**	**1**	23
	D-PAR	4	4	4	3	2	4	4	**1**	4	4	3	2	2	4	4	51
	D-SEQ	4	4	5	4	4	5	4	4	4	5	5	4	5	5	4	67

Rank 1 and 5 correspond to the best and worst performing cases, respectively. We assign the lower rank to both in case of ties. Statistically significant differences are highlighted in bold.

without a neat 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 RQ2 and conclude that LLM-generated test cases can be effectively employed for doing policy

search with GP, given that an appropriate curriculum is chosen. Namely, scheduling the evaluation on multiple test cases in parallel outperforms sequential scheduling.

Curricula Comparison. Another interesting analysis concerns the comparison between the increasing-difficulties curricula—C-PAR, C-SEQ, and C-GRAD—and the difficult-only curricula. Within the sequential scheduling, there seem to be only minor differences, as both C-SEQ and D-SEQ lead to poor evolutionary performance. In fact, the only noticeable distinction between curriculum-based and difficult-only runs is that curriculum-based sequential runs achieved higher fitness on easier levels initially. However, this advantage dissipated when agents encountered more difficult levels, with performance regressing to levels similar to those in the difficult-only runs.

Conversely, examining the parallel scheduling, the differences between C-PAR and D-PAR are well-noticeable and worth discussing. 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, its evolution appears slightly different for the two GGP representations, although in both cases there is actual learning occurring. In particular, for LGP we notice a rather fast evolution at the beginning, which is followed by a plateau, meaning the population has converged to some local optimum. Conversely, for CGP, we observe a continuous fitness increase, indicating a population less prone to converging on local optima and more incline to continue learning throughout the entire evolutionary optimization.

Concerning the gradual scheduling, C-GRAD, we note a behavior that is remarkably similar to that of C-PAR, in that we observe learning intertwined by performance drops when the test cases change. Interestingly, for LGP a more gradual scheduling does not seem particularly beneficial with respect to C-PAR, whereas for CGP we note a starker difference in final performance between C-PAR and C-GRAD, in favor of the second.

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. In addition, for one of the two representations a more gradual CL approach has further boosted the benefits offered by CL.

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 the two bottom plots of Figure 15. These results complement 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 for both GGP representations, and the neat superiority of the parallel and gradual scheduling with respect to the sequential one. However, the p-values resulting from the comparisons of C-PAR and D-PAR, C-PAR and C-GRAD, and D-PAR and C-GRAD only highlight statistical significance for D-PAR vs. C-GRAD for the LGP representation. Hence, in conclusion C-GRAD seems to be the best option, yet with no widespread 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.

GGP Representation Comparison. Moving to our third research question, we analyze the plots of Figure 15 under a new light, focusing on the differences among GGP representations. As already

anticipated above, the main differences are visible for the parallel and gradual schedules, while both representations achieve similar results within the sequential schedules.

The first noticeable difference regards the behavior on D-PAR, where CGP surprisingly achieved a higher average fitness compared to C-PAR. This suggests that exposing agents to challenging environments from the outset may, in some cases, lead to superior performance.

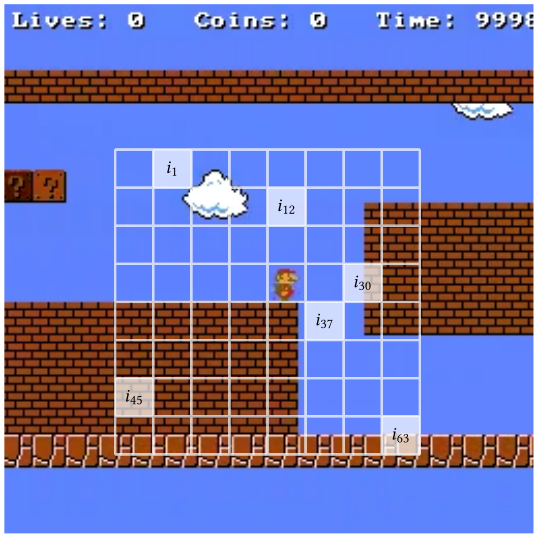
Second, focusing more on C-PAR and C-GRAD, we can note that while CGP policies generally outperformed their LGP counterparts in the intermediate stages of the curriculum, this did not hold at the highest difficulty level, i.e., at the end of evolution. This indicates that the CGP evolutions may be more sensitive to extreme difficulty spikes than the LGP ones.

In summary, our comparative analysis of the representations revealed some inconsistencies in performance during the evolutionary process. However, these differences tended to diminish by the end of optimization, as shown in the lower plots of Figure 15. While we did not formally test for statistical significance, the observed trends suggest that despite potential differences in search dynamics between the two representations, both approaches tend to converge to similarly strong results when given sufficient computational resources. Therefore, we can tentatively answer RQ3 in the affirmative, as the outcomes appear broadly consistent across representations.

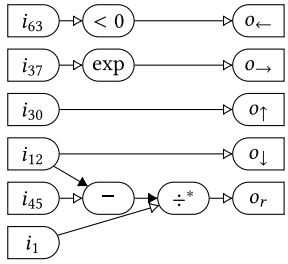
Example of a CGP Policy. In Figure 16, we analyze a policy obtained with CGP to play Super Mario. This visualization provides insight into the structure and rationale of the evolved policy by linking its graph-based genotype to its symbolic behavior in a specific test case. At the top, we show a sample environment frame, with the inputs to the agent highlighted. These inputs correspond to local tiles surrounding the position of Mario and encode information such as nearby obstacles, enemies, and open paths. The center of the figure presents the evolved policy in its native CGP representation as a DAG, and the same policy is also shown in symbolic form below. A portion of the genotype $\boldsymbol{\theta}$ contributing to the selected output (e.g., the "run" action o_r) is highlighted, and we show how the graph and symbolic expression are derived from the integer-based genotype through a decoding process. In this example, the policy always favors moving right, as the exponential term in $o_{\rightarrow}$ ensures it is consistently greater than zero. This is coherent with the game as the progress in Super Mario is primarily achieved by advancing to the right. The agent also includes a condition that activates the jump action ($o_{\uparrow}$) when there is an obstacle directly ahead, suggesting the learned behavior of avoiding gaps or enemies. Furthermore, the down action ($o_{\downarrow}$) is triggered when a tile above is occupied, allowing the agent to drop down if blocked from above. The expression for the "run" output is given by $o_r = \frac{i_1}{i_{45} - i_{37}}$, where i_1 corresponds to the upper-left tile, i_{45} to the bottom-left, and i_{37} to the bottom-right of Mario's position. This expression encourages running when the upper-left tile is occupied—e.g., Mario is under a ceiling or platform. In this configuration, moving forward is generally safe and efficient.

Generalization. Proceeding to the generalization results, we display the distribution of the performance obtained by the best evolved individuals on the eight new LLM-generated test cases, grouped by difficulty (see the titles of the plots) and by representation (see the label on the right side of the rightmost plots), in Figure 17; we report the results of statistical tests in Table 1. Testing the evolved policies on unseen test cases is fundamental to assess if they overfit or they actually learned the game play.

We analyze the results starting with the sequential scheduling. Across both GGP representations, we observe no significant differences between C-SEQ and D-SEQ (see Table 1), as already noted for the results during and at the end of evolution. In addition, we can note that both C-SEQ and D-SEQ are generally the worst performing scenarios compared to C-PAR, C-GRAD, and D-PAR, with significantly worse results in 36/48 cases. Thus, we can further confirm the inferiority of the sequential scheduling.

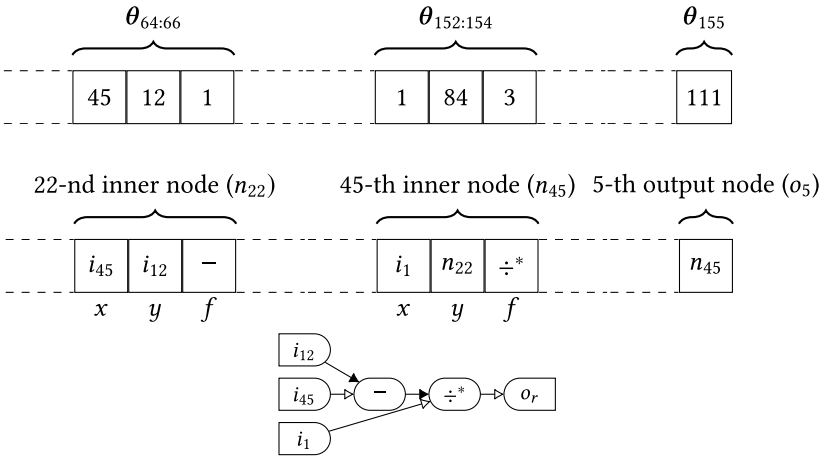


(a) Super Mario Bros test case.



$$o_{\leftarrow} = 1 \text{ if } i_{63} < 0, \text{ else } 0$$
$$o_{\rightarrow} = \exp(i_{37})$$
$$o_{\downarrow} = i_{12}$$
$$o_{\uparrow} = i_{30}$$
$$o_r = \frac{i_1}{i45 - i37}$$

(b) CGP policy.



$$o_r = o_5 = n_{45} = i_1 \div^* n_{22} = i_1 \div^* (i_{45} - i_{12}) = \frac{i_1}{i_{45} - i_{12}}$$

(c) Portion of the genotype $\boldsymbol{\theta}$ with a sketch of the encoding process.

Fig. 16. Representation of a sample Super Mario Bros test case (i.e., environment) with the inputs to the policy highlighted (a), a sample policy found by CGP shown in its original graph form and as formulae (b), and a portion of the genotype encoding it (c). In the graph in (a) and (b), input nodes are shown with rounded angles on the right; and output nodes with rounded angles on the left; for inner nodes with arity 2, the second input is marked with full arrow heads $\rightarrow$; for all inner nodes, the first input is marked with empty arrow heads $\rightarrow$. In figure (c), we show how a portion of the genotype $\boldsymbol{\theta}$ is mapped to the portion of the graph relative to the fifth output o_5, i.e., the run action o_r: in the top row there is the raw genotype $\boldsymbol{\theta}$, in the second row its interpretation with the conversion from integers to pointers (the first $8 \times 8 = 64$ integers refer to inputs, then there are two constants, after that all integers refer to nodes), in the third row there is the corresponding portion of the graph policy, and finally in the bottom row the corresponding equation for o_r.

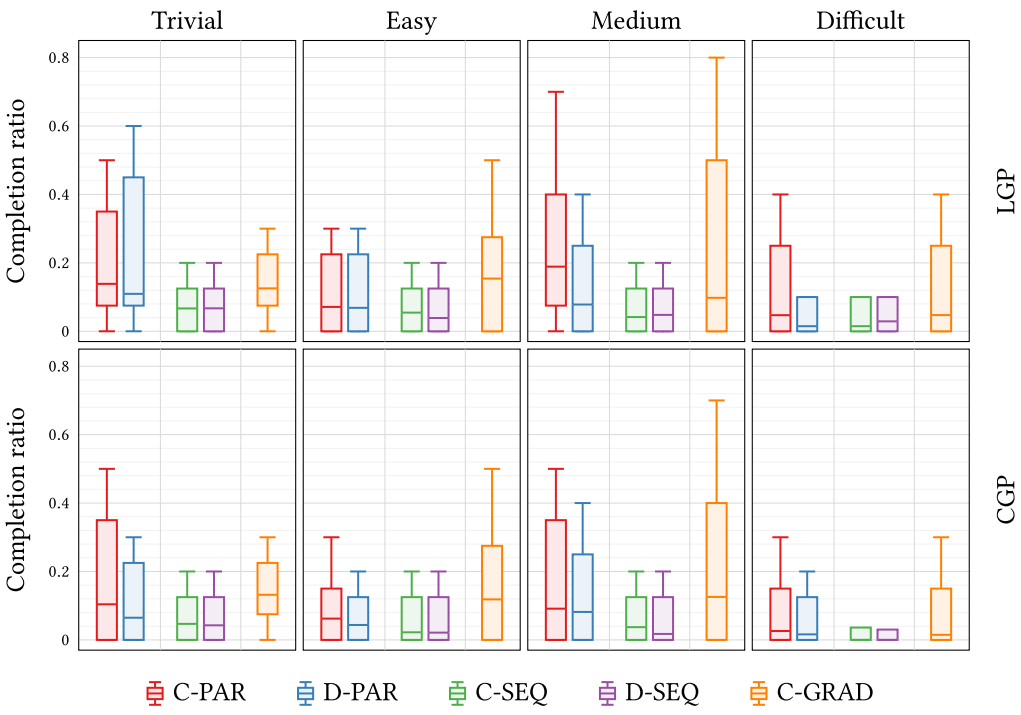


Fig. 17. Distribution of the completion fraction achieved by the 40 best LGP and CGP policies (10 seeds, 4 curricula) on 8 new LLM-generated test cases per difficulty. The statistical analysis of this data is reported in Table 1.

Restricting our analysis to the remaining scenarios, i.e., C-PAR, C-GRAD, and D-PAR, we examine the results starting with the trivial and easy test cases. In these cases the best performing scheduling appears to be C-GRAD—the results are statistically significant for CGP only on the trivial difficulty, and for both representations on the easy difficulty. In addition, for CGP on trivial levels, C-PAR is statistically superior to D-PAR, which further stresses the importance of leveraging CL, especially within this representation. However, for all remaining scenarios, D-PAR is comparable to the considered CL strategies, aligning with previous CL works [Gabor et al., 2019; Wang et al., 2019], 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.

Moving to medium difficulty, we confirm again the importance of CL, as C-GRAD significantly surpasses C-PAR on both representations, with C-PAR also significantly outperforming D-PAR for LGP. These small differences among representations corroborate our previous finding with respect to the different search dynamics of the two representations.

Last, for difficult test cases we note no statistical differences for either GGP representation. In this scenario, it is expected that all policies behave reasonably well, as they have all encountered these types of test cases in training.

To conclude our analysis, we show the evolved policies performance on the preexisting test cases in Table 2. Namely, we report the rank achieved, in median, by the best individual evolved according to each curriculum—C-PAR, C-SEQ, C-GRAD, D-PAR, and D-SEQ—for each test case (one per column), grouped by GGP representation. We do not detail the absolute performance obtained as it is inline with respect to that achieved on the LLM-generated test cases depicted in Figure 17. From this table we can again reason comparatively among the considered curricula, observing that

either C-PAR or C-GRAD generally outperform all other curricula, always ranking first or second, confirming most of our previous findings. Following 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.

4.3 Discussion

The analysis in Section 4.2 indicates that LLM-generated test cases enable GP to search effectively for agent policies in both two-player and single-player games. The policies evolved by GP 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 evolution performed by GP 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 evolved policy 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.

Our results also show that not all the LLMs are equally good at generating test cases (a) of actually different difficulty and (b) actually challenging. For the Connect-4 game, while all LLMs were able to generate opponents which can win against trivial (random) players, Llama-generated opponents did not significantly differ from this point of view, while being, for the largest model (Lllama-3.1-Instruct-405B), effective against ChatGPT-4 opponents. However, we found a general trend in the extrinsic results, regardless of the LLM being employed. We believe this trend supports our hypothesis that LLMs can be indeed used as assistants for building curricula to employ within GP optimizations.

5 Conclusion, Limitations, and Future Work

We devised an approach for exploiting LLMs for generating test cases for evolutionary policy search performed through GP. Moreover, we proposed to schedule LLM-generated test cases according to the principles of 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. For the two-player board game, for which we could use general-purpose LLMs, we compared different models focusing on their ability to generate diverse and challenging test cases. Experimentally, we showed the importance of gradually increasing the difficulty of the test cases during learning for achieving well-performing and robust GP agents. We also assessed the robustness of the approach with respect to the representation of the policy for the playing agent: we employed linear programs using LGP and with graphs using CGP.

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 and for which the best practices are not necessarily the same across different LLMs. Guo et al. [2023] 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 [Bender et al., 2021]. Moreover, closed source LLMs which can only be accessed via dedicated interfaces might change their behavior in a non-predictable way due to back-end updates. Another point to consider is the *familiarity* of the LLM with the problem domain and potential biases from its training [Huang et al., 2023; Yeh et al., 2023]. For unfamiliar domains, breaking them into known segments or providing wider context can help the LLM generate more effective outputs [Song et al., 2023]. Moreover, MarioGPT and similar studies, such as Todd et al. [2023], demonstrate that fine-tuning LLMs can also be effective in tackling the lack of domain knowledge. As noted by Li et al. [2023], this approach demands considerable computational effort and training data.

Future research should assess the generality of our approach by exploring more complex multi-player games like Othello [Li et al., 2022], 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. [2023] developed a fine-tuned LLM 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 [Guo et al., 2018] could enable a co-evolutionary setup [Thompson, 2014], allowing prompts and agents to evolve together to improve the overall performance.

References

Mohamed F. Abdelsadek. n.d. *Using Genetic Programming to Evolve a Connect-4 Game Player*. Department of Computer Science, Columbia University. Retrieved from https://www.cs.columbia.edu/~evs/ml/MLfinalproj/mohamed/connect4.html

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 technical report. arXiv:2303.08774. Retrieved from https://arxiv.org/abs/2303.08774

Louis Victor Allis. 1994. *Searching for Solutions in Games and Artificial Intelligence*. Ponsen & Looijen Wageningen.

Ryan Amaral, Alexandru Ianta, Caleidgh Bayer, Robert J. Smith, and Malcolm I. Heywood. 2022. Benchmarking genetic programming in a multi-action reinforcement learning locomotion task. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 522–525.

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*. Antonio J. Tallón-Ballesteros (Ed.), IOS Press, 390–396.

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.

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.

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.

Markus F. Brameier and Wolfgang Banzhaf. 2007. *Basic Concepts of Linear Genetic Programming*. Springer.

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. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 33, 1877–1901.

Youngjin Chae and Thomas Davidson. 2023. Large language models for text classification: From zero-shot learning to fine-tuning. *Open Science Foundation* (2023). DOI: https://doi.org/10.31235/osf.io/sthwk_v2

Tim Cofala, Lars Elend, and Oliver Kramer. 2020. Tournament selection improves cartesian genetic programming for Atari games. In *Proceedings of the European Symposium on Artificial Neural Networks*, 345–350.

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.

Mayank Dabas, Nishthavan Dahiya, and Pratish Pushparaj. 2022. Solving Connect 4 using artificial intelligence. In *Proceedings of the International Conference on Innovative Computing and Communications (ICICC '21)*, Vol. 1. Springer, 727–735.

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

Camilo De La Torre, Kévin Cortacero, Sylvain Cussat-Blanc, and Dennis Wilson. 2024. Multimodal adaptive graph evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 499–502.

Camilo De La Torre, Giorgia Nadizar, Yuri Lavinas, Hervé Luga, Dennis Wilson, and Sylvain Cussat-Blanc. 2025. Evolution of inherently interpretable visual control policies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '25)*. Association for Computing Machinery, New York, NY, USA, 358–367. DOI: https://doi.org/10.1145/3712256.3726332

Andrea De Lorenzo, Alberto Bartoli, Mauro Castelli, Eric Medvet, and Bing Xue. 2020. Genetic programming in the twenty-first century: A bibliometric and content-based analysis from both sides of the fence. *Genetic Programming and Evolvable Machines* 21 (2020), 181–204.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 Herd of models. arXiv:2407.21783. Retrieved from https://arxiv.org/abs/2407.21783

Stefan Edelkamp and Peter Kissmann. 2008. Symbolic classification of general two-player games. In *Proceedings of the Advances in Artificial Intelligence: 31st Annual German Conference on AI (KI '08)*. Springer, 185–192.

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

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:2405.15568. Retrieved from https://arxiv.org/abs/2405.15568

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. 2017. Reverse curriculum generation for reinforcement learning. In *Proceedings of the Conference on Robot Learning*. PMLR, 482–495.

Léo Françoso Dal Piccol Sotto, Paul Kaufmann, Timothy Atkinson, Roman Kalkreuth, and Márcio Porto Basgalupp. 2021. Graph representations in genetic programming. *Genetic Programming and Evolvable Machines* 22, 4 (2021), 607–636.

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.

Imran Ghory. 2004. *Reinforcement Learning in Board Games*. Technical Report 105. Department of Computer Science, University of Bristol.

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 *Proceedings of the International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 765–779.

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:2309.08532. Retrieved from https://arxiv.org/abs/2309.08532

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.

Cheng He, Ye Tian, and Zhichao Lu. 2025. Artificial evolutionary intelligence (AEI): Evolutionary computation evolves with large language models. *Journal of Membrane Computing* 7 (2025), 135–152.

Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. 2024. Evolving code with a large language model. *Genetic Programming and Evolvable Machines* 25, 2 (2024), 21.

Dong Huang, Qingwen Bu, Jie Zhang, Xiaofei Xie, Junjie Chen, and Heming Cui. 2023. Bias assessment and mitigation in LLM-based code generation. arXiv:2309.14345. Retrieved from https://arxiv.org/abs/2309.14345

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.

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.

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

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.

Stephen Kelly and Malcolm I. Heywood. 2017a. Emergent tangled graph representations for Atari game playing agents. In *Proceedings of the 20th European Conference on Genetic Programming (EuroGP '17)*. Springer, 64–79.

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

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. In *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3889–3896.

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.

John R. Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 2 (1994), 87–112.

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

Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Gloria Pietropolli, and Claudio Zandron. 2023. Inferring P systems from their computing steps: An evolutionary approach. *Swarm and Evolutionary Computation* 76 (2023), 101223.

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:2210.13382. Retrieved from https://arxiv.org/abs/2210.13382

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), e40895.

Yixin Liu, Kejian Shi, Katherine He, Longtian Ye, Alexander Richard Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. 2024. On learning to summarize with large language models as references. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 8639–8656.

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

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

Francesco Marchetti, Gloria Pietropolli, Federico Julian Camerota Verdù, Mauro Castelli, and Edmondo Minisci. 2024. Automatic design of interpretable control laws through parametrized genetic programming with adjoint state method gradient evaluation. *Applied Soft Computing* 159 (2024), 111654.

Nicola Milano and Stefano Nolfi. 2021. Automated curriculum learning for embodied agents a neuroevolutionary approach. *Scientific Reports* 11, 1 (2021), 8985.

Julian Francis Miller. 2020. Cartesian genetic programming: Its status and future. *Genetic Programming and Evolvable Machines* 21 (2020), 129–168.

Julian F. Miller and Peter Thomson. 2000. Cartesian genetic programming. In *Genetic Programming*. Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty (Eds.), Springer, Berlin, 121–132. Retrieved from https://link.springer.com/chapter/10.1007/978-3-540-46239-2_9

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. *ACM Computing Surveys* 56, 2 (2023), 1–40.

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

Giorgia Nadizar, Eric Medvet, and Karine Miras. 2022. On the schedule for morphological development of evolved modular soft robots. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*. Springer, 146–161.

Giorgia Nadizar, Eric Medvet, and Dennis Wilson. 2024a. Searching for a diversity of interpretable graph control policies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 933–941.

Giorgia Nadizar, Eric Medvet, and Dennis G. Wilson. 2024b. Naturally interpretable control policies via graph-based genetic programming. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*. Springer.

Giorgia Nadizar, Eric Medvet, and Dennis G. Wilson. 2025. Enhancing adaptability in embodied agents: A multi-quality-diversity approach. *IEEE Transactions on Evolutionary Computation* (2025). DOI: https://ieeexplore.ieee.org/document/11119655

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.

Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. 2024c. An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. *ACM Transactions on Evolutionary Learning and Optimization* 4, 1 (2024), 1–30.

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.

Sanmit Narvekar, Jivko Sinapov, and Peter Stone. 2017. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2536–2542.

Rijul Nasa, Rishabh Didwania, Shubhranil Maji, and Vipul Kumar. 2018. Alpha-beta pruning in mini-max algorithm—An optimized approach for a Connect-4 game. *International Research Journal of Engineering and Technology* 5, 4 (2018), 1637–1641.

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

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom Mitchell. 2019. Competence-based curriculum learning for neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Jill Burstein, Christy Doran, and Thamar Solorio (Eds.), Association for Computational Linguistics, Minneapolis, Minnesota, 1162–1172. DOI: https://doi.org/10.18653/v1/N19-1119

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

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. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 35, 36479–36494.

Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. 2019. Data parameters: A new family of parameters for learning a differentiable curriculum. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol 32.

Marvin Oliver Schneider and J. L. Garcia Rosa. 2002. Neural connect 4—A connectionist approach to the game. In *Proceedings of the VII Brazilian Symposium on Neural Networks (SBRN '02)*. IEEE, 236–241.

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.

Yehonatan Shichel, Eran Ziserman, and Moshe Sipper. 2005. GP-Robocode: Using genetic programming to evolve Robocode players. In *Proceedings of the European Conference on Genetic Programming*. Springer, 143–154.

Peer Sommerlund. 1996. Artificial neural nets applied to strategic games. *Unpublished, Last Access* 5 (1996), 12.

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.

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.

Martin Stenmark. 2005. *Synthesizing Board Evaluation Functions for Connect-4 Using Machine Learning Techniques*. Master's thesis. Østfold University College, Norway.

Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. 2024. MarioGPT: Open-ended Text2Level generation through large language models. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 36.

Yi Tay, Shuohang Wang, Anh Tuan Luu, 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. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4922–4931.

Markus Thill, Patrick Koch, and Wolfgang Konen. 2012. Reinforcement learning with n-tuples on the game connect-4. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature-PPSN XII, Part I*. Springer, 184–194.

John N. Thompson. 2014. *Interaction and Coevolution*. University of Chicago Press.

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.

Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. 2010. The 2009 Mario AI competition. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 1–8.

Julian Togelius, Sergey Karakovskiy, Koutník Jan, and Schmidhuber Jurgen. 2009. Super Mario evolution. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 156–161.

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

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 30.

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

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.

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.

Daphna Weinshall, Gad Cohen, and Dan Amir. 2018. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 5238–5246.

Dennis G. Wilson, Sylvain Cussat-Blanc, Hervé Luga, and Julian F. Miller. 2018a. Evolving simple programs for playing Atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 229–236.

Dennis G. Wilson, Julian F. Miller, Sylvain Cussat-Blanc, and Hervé Luga. 2018b. Positional cartesian genetic programming. arXiv:1810.04119. Retrieved from https://arxiv.org/abs/1810.04119

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:2401.10034. Retrieved from https://arxiv.org/abs/2401.10034

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 '23)*, 292–299.

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. Retrieved from https://arxiv.org/abs/2403.12014

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.

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:2309.01219. Retrieved from https://arxiv.org/abs/2309.01219

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:2303.18223. Retrieved from https://arxiv.org/abs/2303.18223

Appendix

LLM Prompting for Connect-4 Policies

In our study, we employed a systematic approach to prompt the Llama-3.1 LLMs to generate Connect-4 policies of varying difficulties. Our approach centered on crafting simple but effective prompts, paired with the previous level difficulty policy as an example, to guide the LLMs in generating functional game policies that increase in difficulty. Prompt engineering techniques play an important role in optimizing interactions with LLMs like those used for Connect-4 policies. It is important to provide clear and specific instructions within the prompt, without restricting the creativity of the model too much. This involves stating the desired outcome, such as generating a Connect-4 policy of a particular difficulty level, and outlining any specific requirements, constraints, or representational details. Another common technique to helping models generate effective outputs is to give the model an example of the desired output (known as One-Shot prompting or Few-Shot prompting if more than one example is provided).

Our prompts were structured to help the LLMs create working policies at different difficulty levels. We began each prompt with explicit instructions about the task, specifying the need for a Connect-4 policy and the desired difficulty level. We utilized One-Shot prompting by including the policy the LLM generated for the previous difficulty level as a reference point for the LLM (to generate the Easy level policies, a manually crafted random player was provided as the reference point) and described the expected board representation. For the difficulty levels above easy we also

prompted the LLM to consider specific strategic elements of Connect-4, such as blocking opponent wins and setting up win conditions. This is to help ensure the more difficult strategies don't regress in performance by prioritizing new strategies over making a winning move or preventing an opponent from winning. Examples of the prompts we used are shown below:

Prompt for Generating an Easy Difficulty Policy.

Given the following random player for playing connect-4, update it to make an easy policy to play against that helps a player learn good strategies. Assume the board state is a 2d numpy array where 0 represents an empty slot, 1 represents the red player, and −1 represents the yellow player. Assume each move in the possible_moves are just a single integer indicating the column to play in and assume the board_size is one integer:

```
class RandomPolicy(object):
"""Random policy for Connect 4."""

def __init__(self):
    self.env = None

def reset(self, env):
    if hasattr(env, 'env'):
        self.env = env.env
    else:
        self.env = env

def get_action(self, obs):
    possible_moves = self.env.possible_moves
    board_state = self.env.board_state.copy()
    board_size = self.env.board_size
    player_turn = self.env.player_turn

    ix = random.randint(0, len(possible_moves)-1)
    action = possible_moves[ix]
    return action
```

Prompt for Generating a Medium Difficulty Policy.

Given the following easy difficulty Policy for playing connect-4, update it to make a medium policy to play against that helps a player learn good strategies. Assume the board state is a 2d numpy array where 0 represents an empty slot, 1 represents the red player, and −1 represents the yellow player. Assume each move in the possible_moves are just a single integer indicating the column to play in and assume the board_size is one integer. Be sure to prioritize winning the game and blocking the opponent from winning:

```
class EasyPolicy(object):
"""Easy policy for Connect 4."""

"""
The easy difficulty policy the LLM generated would be inserted here.
"""
```

Prompt for Generating a Hard Difficulty Policy.

Given the following medium difficulty Policy for playing connect-4, update it to make a hard policy to play against that helps a player learn good strategies. Assume the board state is a 2d numpy array where 0 represents an empty slot, 1 represents the red player, and −1 represents the yellow player. Assume each move in the possible_moves are just a single integer indicating the column to play in and assume the board_size is one integer. Be sure to prioritize winning the game and blocking the opponent from winning:

```
class MediumPolicy(object):
"""Medium policy for Connect 4."""

"""
The medium difficulty policy the LLM generated would be inserted here.
"""
```

Prompt for Generating an Expert Difficulty Policy.

Given the following hard Policy for playing connect-4, update it to make an even more difficult policy to play against, making sure to prioritize both winning the game and blocking the opponent from winning. Assume the board state is a 2d numpy array where 0 represents an empty slot, 1 represents the red player, and −1 represents the yellow player. Assume each move in the possible_moves are just a single integer indicating the column to play in and assume the board_size is one integer:

```
class HardPolicy(object):
"""Hard policy for Connect 4."""

"""
The hard difficulty policy the LLM generated would be inserted here.
"""
```

The generated policies were manually copied from the model output and tweaked slightly by updating the policy names to represent which model generated them (ex: EasyPolicy405B), then each policy was tested for functionality by having it play 1000 games against the manually crafted random player (see Figure 3). Occasionally a LLM generated policy would include a call to a created function with no implementation provided for that function. In this case, we prompted the model to provide an implementation of the function and added the outputted function to the policy. Additionally, LLM generated policies were sometimes found to be unable to beat the random player, in which case the LLM was re-prompted again using the same prompt with an addendum that the outputted policy be able to beat an opponent that plays random moves. Once this testing was complete, the policies were integrated into our experimental framework.

Received 23 December 2024; revised 25 May 2025; accepted 6 October 2025