efficacy, zosurabalpin has been evaluated in two phase I clinical trials[8].

Compounds isolated from nature or the close relatives of such products have long dominated antibacterial drug discovery, because many such compounds have evolved the ability to kill bacteria. Indeed, only a handful of antibiotic classes — most notably sulfonamides, fluoroquinolones and oxazolidinones — are synthetic[3]. As such, the discovery and development of zosurabalpin is particularly notable, and it demonstrates that examining alternative sources of possible antibiotics beyond those that are usually tested — including compounds from sources such as the MCP collection, which have a higher molecular weight than is typical for antibiotics — can be fruitful. This has also been demonstrated in other work with DNA-encoded molecular 'libraries'[9].

Drug discovery that targets harmful Gram-negative bacteria is a long-standing challenge owing to difficulties in getting molecules to cross the bacterial membranes to reach targets in the cytoplasm. Successful compounds typically must possess a certain combination of chemical characteristics[10,11]. It therefore makes sense to pursue antibiotics that engage targets on the outside of the cell or in the space between the inner and outer membranes, termed the periplasm. $LptB_2FGC$ resides in the periplasm, so zosurabalpin does not need to reach the bacterial cytoplasm.

This success with zosurabalpin mirrors that of other newly discovered antibiotics, such as darobactin[12] and dynobactin[13], which are also high-molecular-weight compounds that need to reach targets outside the cytoplasm. Non-cytoplasmic targets can probably be engaged by more-chemically-diverse compounds (with respect to their size and shape) than can cytoplasmic targets.

With nearly all antibiotics, mutations (often in the target) arise that lead to antibiotic resistance. Indeed, mutations in genes encoding components of $LptB_2FGC$ in zosurabalpin-resistant CRAB identified in the lab result in a striking decrease in zosurabalpin's ability to kill the bacterium. In one case, a single-nucleotide mutation in the sequence encoding the $LptB_2FGC$ complex resulted in a more than 256-fold decrease in antibiotic activity. The unusual mode of action of zosurabalpin requires LPS, and this foreshadows another potential limitation because *A. baumannii* is unusual in that it does not need LPS to be viable[14]. The bacterium can halt LPS synthesis if necessary for survival — potentially rendering zosurabalpin ineffective against LPS-deficient *A. baumannii*. However, such a change would attenuate bacterial virulence[15], and it remains to be determined whether this type of antibiotic-resistance mechanism will be observed in the clinic.

This discovery opens the door to targeting the LPS-transport system of other problematic Gram-negative bacteria, such as *Pseudomonas aeruginosa*, *Klebsiella pneumoniae* and *Escherichia coli*. In addition, there is now an appreciation that disruption of normal gut microbes (the microbiome) is deleterious to human health; such a disturbance is a consequence of essentially all antibiotics used in the clinic. Given zosurabalpin's high specificity for *A. baumannii*, it might be a microbiome-sparing antibiotic. The movement towards bacterium-specific antibiotics is a new development, and one that can be facilitated by diagnostics that can rapidly identify specific harmful bacteria in infected individuals[3]. Given that zosurabalpin is already being tested in clinical trials, the future looks promising, with the possibility of a new antibiotic class being finally on the horizon for invasive CRAB infections.

**Morgan K. Gugger** and **Paul J. Hergenrother** are in the Department of Chemistry, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.
e-mail: hergenro@illinois.edu

1. Antimicrobial Resistance Collaborators. *Lancet* **399**, 629–655 (2022).
2. Silver, L. L. *Bioorg. Med. Chem.* **24**, 6379–6389 (2016).
3. Lewis, K. *Cell* **181**, 29–45 (2020).
4. Du, X. *et al. Am. J. Infect. Control* **47**, 1140–1145 (2019).
5. Zampaloni, C. *et al. Nature* **625**, 566–571 (2024).
6. Pahil, K. S. *et al. Nature* **625**, 572–577 (2024).
7. Theuretzbacher, U., Blasco, B., Duffey, M. & Piddock, L. J. V. *Nature Rev. Drug Discov.* **22**, 957–975 (2023).
8. Guenther, A. *et al. Open Forum Infect. Dis.* **10**, ofad500.1749 (2023).
9. Ryan, M. D. *et al. J. Med. Chem.* **64**, 14377–14425 (2021).
10. Richter, M. F. *et al. Nature* **545**, 299–304 (2017).
11. Geddes, E. J. *et al. Nature* **624**, 145–153 (2023).
12. Imai, Y. *et al. Nature* **576**, 459–464 (2019).
13. Miller, R. D. *et al. Nature Microbiol.* **7**, 1661–1672 (2022).
14. Henry, R. *et al. Antimicrob. Agents Chemother.* **56**, 59–69 (2012).
15. Beceiro, A. *et al. Antimicrob. Agents. Chemother.* **58**, 518–526 (2014).

**The authors declare no competing interests.**
**This article was published online on 3 January 2024.**

Computer science

# Large language models help programs to evolve

**Jean-Baptiste Mouret**

A branch of computer science known as genetic programming has been given a boost with the application of large language models that are trained on the combined intuition of the world's programmers. **See p.468**

Although machines outperform humans at many tasks, from manufacturing to playing games[1,2], they are commonly considered incapable of creating or inventing things. But this is changing: over the past three years, technology companies have been releasing artificial intelligence (AI) programs that can draw or write with impressive creativity. Scientific discovery might be the next target, as computers start to uncover knowledge that has eluded scientists. On page 468, Romera-Paredes *et al.*[3] report an autonomous mathematical discovery for which AI was used — not to check a proof or to execute tedious computations, but to solve open problems. This proof of concept is likely to be followed by other programs like it, as software becomes a creative contributor to scientific discoveries.

Romera-Paredes and colleagues' work is the latest step in a long line of research that attempts to create programs automatically by taking inspiration from biological evolution, a field called genetic programming[4]. The process starts with running many random programs to find out how well each one can solve a target problem. The best programs are then selected, copied and randomly modified, in a manner that is similar to genetic variation. The process then begins again with these modified programs, which are selected and modified until one program solves the problem adequately.

The key question in genetic programming is how to represent programs so that they can be modified easily, but meaningfully, by random variation. In other words, what is the 'DNA' of a computer program? For instance, adding random letters to a program written in the Python scripting language is unlikely to result in a program that follows Python syntax, which means that the vast majority of modified programs cannot be executed by the computer, and are therefore useless.

To approach this problem, genetic-programming researchers have taken inspiration from compilers, which are programs that transform text written in a programming language into code that a computer can

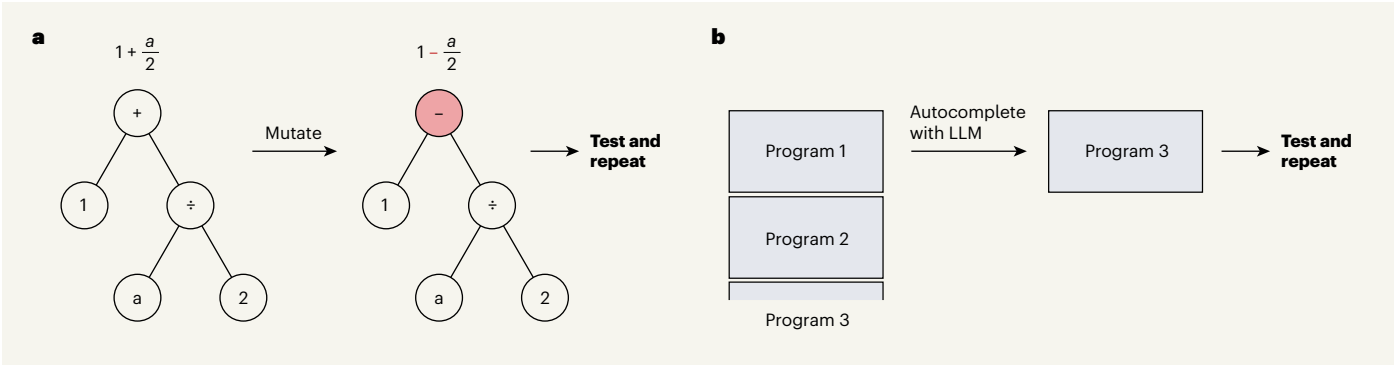


**Figure 1 | Genetic programming with large language models.** Genetic programming is a computer-science approach that 'mutates' code, one variation at a time. **a**, A computer program can be represented with an abstract 'tree' structure. In this representation, genetic programming involves changing one node in the tree at random, testing to see whether the change has improved the program and then repeating that process on the highest-performing programs. **b**, Instead of making changes to an abstract tree, Romera-Paredes *et al.*[3] devised a genetic-programming method that involves taking two programs, concatenating them, and asking a large language model (LLM) to autocomplete a third program using the concatenated pair as a prompt. This approach enabled the authors to generate variants of programs that are likely to be meaningful in their specific context.

interpret. Compilers represent programs in the computer's memory with an abstract 'tree' structure. Using this representation, a genetic-programming system 'mutates' a program by randomly changing one node in the tree to a different value (Fig. 1a). For example, a mutation might replace a plus sign with a minus sign, or the number 2 with a 3. Similarly, the system mixes two programs by exchanging randomly chosen sub-trees from the two separate programs.

This approach has given rise to many fascinating results. For the past 20 years, a competition has been held to showcase how evolution-inspired algorithms measure up to those made by humans (see human-competitive.org/awards). One example is a program called Eureqa, which automatically finds the equations of motion for classic dynamical systems, such as a swinging pendulum, using data alone[5]. Another example is the GenProg system[6], which provides a way of automatically repairing existing open-source programs – and even earned its developers money doing so.

Nevertheless, genetic programming has so far been successful only for equations or short sections of programs, typically comprising a dozen lines. The reason for this is that programming is hard. It requires the right syntax and intuition about what could work. And every change requires the programmer to take the context of the rest of the program into account. In general, random variations in syntax trees are rarely meaningful and the genetic-programming process yields results only through millions of repetitions.

This situation is changing with large language models (LLMs). These large neural networks are trained to predict the next words in a line of code for a given context (known as a prompt), after having been fed millions of lines of code. This large body of examples gives the model an 'intuition' about what a typical programmer would write in the same situation. LLMs that are specialized in code generation are used daily by many programmers in the same way that people use the autocomplete function available in many apps – the model guesses the most likely text to follow each piece of code, so there is no need to type it or to memorize the function names.

LLMs are game-changing tools for programmers, but they might also be the missing piece of the genetic-programming puzzle: they embed the knowledge of thousands of programmers to generate meaningful code for a given context. Instead of replacing random parts of a syntax tree, an LLM can generate a variation of a program written in a standard programming language, such as Python. To do so, a simple, but powerful, approach is to select two programs, concatenate them, and ask the LLM to complete the program using the concatenated pair as a prompt – resulting in the generation of a third program (Fig. 1b). The result will probably have valid syntax and be meaningful in its context. However, it might not be exact or optimal. This is why the process must be iterated, by selecting the highest-performing programs, generating variations using the LLM and then testing these variations.

> **"Genetic programming has so far been successful only for equations or short sections of programs."**

Romera-Paredes *et al.* used this fresh approach to genetic programming to find ways of solving mathematical problems in optimization and geometry that were better than the best attempts of human programmers. The authors' system shows promise, but it still requires guidance in the form of an 'evaluate function', which nudges the model in the most productive direction. A more direct way would be to use a 'validate function', which determines when the problem has been solved. The authors' evaluate function is similar to a school test that has been designed to reward learning by striking a balance between easy and difficult questions.

Using automatic programmers, such as the one developed by Romera-Paredes and colleagues, will require the same level of careful judgement in crafting the right tests. However, the authors' innovation demonstrates the power and potential of using LLMs to write creative programs for solving problems, and this advance is part of a developing success story[7,8]. As the problems intended for these models become ever more relevant, it's clear that LLMs will breathe new life into genetic programming.

**Jean-Baptiste Mouret** is at the French National Institute for Research in Digital Science and Technology (INRIA), the University of Lorraine, and the French national research agency (CNRS), 54000 Nancy, France.
e-mail: jean-baptiste.mouret@inria.fr

1. Silver, D. *et al. Nature* **529**, 484–489 (2016).
2. Hsu, F.-H. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion* (Princeton Univ. Press, 2002).
3. Romera-Paredes, B. *et al. Nature* **625**, 468–475 (2024).
4. Koza, J. R. *Genetic Programming* (MIT, 1992).
5. Schmidt, M. & Lipson, H. *Science* **324**, 81–85 (2009).
6. Le Goues, C., Dewey-Vogt, M., Forrest, S. A. & Weimer, W. R. In *Proc. 34th International Conference on Software Engineering* 3–13 (IEEE, 2012).
7. Lehman, J. *et al.* in *Handbook of Evolutionary Machine Learning* (eds Banzhaf, W. *et al.*) Ch. 11 (Springer Nature Singapore, 2024).
8. Meyerson, E. *et al.* Preprint at https://arxiv.org/abs/2302.12170 (2023).

**The author declares no competing interests.**