Sustaining Evolution for Shallow Embodied Intelligence

W. B. Langdon and Daniel Hulme

Department of Computer Science, University College London, Gower Street, WC1E 6BT, UK

E-mail: w.langdon@cs.ucl.ac.uk, daniel@satalia.com

Abstract. Lenski's experiments with E. Coli show Biology can sustain continual evolutionary improvement. However long term evolutionary experiments (LTEE) with evolutionary computing find that information theory's failed disruption propagation (FDP) in deeply nested genetic programming (GP) hierarchies can greatly slow adaptation. We propose that researchers aiming at embodied artificial intelligence should control software robustness by using porous high surface area geometrical architectures, perhaps composed of many shallow mangrove like tree structures intimately linked to their data rich fitness environment.

1. Introduction

Rich Lenski has shown in biology evolution can retain its ability for continued change even after 75 000 generations [1, 2]. (Homo Sapiens appears to be about 9300 generations old [3].) He started the long term evolutionary experiment (LTEE) in 1998 and it is still running. We have shown genetic programming (GP) [4, 5] can do similarly when run for a million generations and experiments can be run in days or weeks rather than years [6, 7]. However, information theory (Section 3) explains why the rate of fitness improvement falls as very deep GP trees become resistant to crossover. Mutation testing shows that human written software can also be resilient to many source code changes and that there is a tendency for deeply nested code to be more robust [8].

There are examples of software systems that exceed a billion lines of source code. Information theory's failed disruption propagation (FDP) [9] helps to explain why maintaining, testing and debugging deeply nested codebases is hard and why software companies prefer unit testing of modules (each of which is typically only shallowly nested) rather than system testing of complete functional hierarchies. There is already SBSE [10] work on automatically optimising test oracles [11]. FDP suggests software systems aught to be built with many densely packed test agents so that disruption caused by bugs has little distance to travel before being discovered by an oracle.

For evolutionary computing and artificial life experiments aiming for sustained innovation, we propose the use of porous high surface area architectures composed of many small trees which are intimate with their environment (see Figure 10, page 11). For continuous innovative evolution the fitness function needs to be able to measure on average if genetic changes are good or not. That is, they must have made a difference. This means we must overcome robustness, without introducing chaos. We suggest this might be met by systems where the bulk of the code remains close to the fitness environment and the disruption caused by most mutations and crossovers has only a short distance to propagate in order to have a measurable fitness impact.

The interested reader can find Koza's early experiments and introduction to genetic programming in his book [4]. Whilst the Poli's "field guide" [5] is specifically aimed at students and includes some more recent work.

In evolutionary computing "complexity analysis" has several meanings. For example it can mean traditional computer science (Big O) complexity analysis, which aims at deriving formal mathematics saying important things about solving particular problems such as how long it will take on average and how runtime scales with the dimensions of the problem [12, 13]. Complexity can also be in terms of information theory. With complex solutions containing more information. Although strictly not computable, Kolmogorov defines the complexity of a string as the size of shortest program able to reproduce the string. Vitanyi has shown in practise it can be estimated using every day compression tools [14, 15, 16, 17]. Finally the size of solutions is often taken as related to their complexity. However, without counter measures there is a tendency for solution size to increase [18, 19]. Sometimes it is assumed that smaller solutions will be more human comprehensible and so preferred and nowadays there are a variety of sophisticated anti-bloat tools [20, 21]. Although size can be measured and Kolmogorov complexity can be estimated for GP solutions [22], the information needed to reproduce the evolved program can fail to usefully capture our notion of the complexity of the solution. For example, programs in a GP population can all be different (i.e. their genotypes are not the same) and yet their behaviour (phenotypes) are the same [7].

2. Sustaining "Thin" Evolutionary Machine Learning

One lesson we can take from the recent successes of artificial intelligence (AI) is that huge AI systems can be created automatically when they are based on learning. That is, when we automatically extract knowledge from training data, rather than manually encoding knowledge into a computer based repository. We suggest in the hunt for the next level artificial intelligence or machine consciousness [23], we should consider systems which, instead of assembling all the training data and learning everything the system can do in one huge bout of training, we should investigate systems which continually learn.

The next section gives a quick introduction to information theory and applying it to software, showing that entropy loss leads to failed disruption propagation (FDP) and so to robust software that is error tolerant. Section 4 shows an example of recent measurements of FDP in hand written C, where deeper code tends to be more robust. This is followed in Section 5 by studies of FDP in artificially evolved (genetic programming) code which shows, as expected, deep nested hierarchies are also robust. The discussion (Sections 6 and 7) suggests FDP is not only widespread in existing systems but that future AI systems must take it into account and suggests the adoption of "thin" shallow data rich architectures. The second part of the discussion (Section 7) considers the creativity of evolving systems and the widespread adoption of genetic programming, before we conclude in Section 8 that to demonstrate automatic Embodied Intelligence we need systems that continue to evolve and so recommend architectures that are not too robust but instead permit continuous change. Therefore we should investigate architectures composed of shallow code (i.e. the opposite of deeply nested code) which have a large exposed surface facing a data rich environment.

3. Information Theory and Software

All computer operations are irreversible. Meaning in general given an operation's output we cannot infer its inputs. Figure 1 show this graphically. The left hand side of Figure 1 depicts adding together two 32 bit floating point values to give a single 32 bit output. Because addition is not reversible, given the output, we cannot infer the inputs. For example, $1.0 + 1.0 = 2.0$ and $1.5 + 0.5 = 2.0$, so given the output is 2.0 we cannot say what the two inputs were. That is, we cannot reverse the computation and so information about the inputs has been lost.

We can say how much information there is by calculating the entropy of a distribution of values. For example, the entropy of the input distribution and the entropy of the output distribution. The difference between the two says how much information has been destroyed by the operation. In fact, this can be done not just for a single operation but also for whole programs. The right hand side of Figure 1 contains two plots. The solid line (red) shows the distribution of integer values, which are added together to give the output (dashed blue). In the example, both inputs are digits, i.e., the numbers 0 to 9. Here digits 0 and 1 are more common than the other eight. (The example comes from C source code used in Section 4.) The graph assumes each input is independent of the other and so the output is one of the nineteen integers 0 to 18. The dashed blue plot shows their distribution. The information content (entropy) of either input is 2.88 bits. As they are independent, the information of them together is $2 \times 2.88 = 5.76$ bits. Note the information content (entropy) of the output is 3.75 bits. So this (irreversible) addition has lost $5.76 - 3.75 = 2.01$ bits of information. Entropy loss is inherent in irreversible operations. I.e. all computation loses entropy.

In [9] we consider the informational impact of all forms of disruptions to the smooth execution of software. We treat together all types of disruption, be they coding errors (software bugs), source code changes introduced by mutation testing [24], subtree swaps made by genetic programming crossover, transients due to cosmic rays, EM radiation inserted by a hostile opponent, etc., as long as they change the state of the program during execution. For such a disruption to be effective, that state change has to propagate from its origin to the program's output. If it does not, the program is robust to the error. From an information theory point of view, information about the disruption has to propagate to the output. In nested hierarchies, information loss is progressive. So that even partial information lost cannot be recovered. Hence when information has to pass through many levels of nesting, all information about the disruption may be lost, giving rise to failed disruption propagation (FDP). But as we have seen, because digital computation is not-reversible, entropy (i.e. information) loss is inherent in calculation. So we expect FDP in any deeply nested calculation. In fact, FDP is common even in languages with side-effect (e.g. global variables in C/C++) that might allow information to by-pass function call nesting. Thus in non-trivial software small transients tend not to have any knock on consequences (Figure 2). Leading ordinary software to be robust to many errors [25].

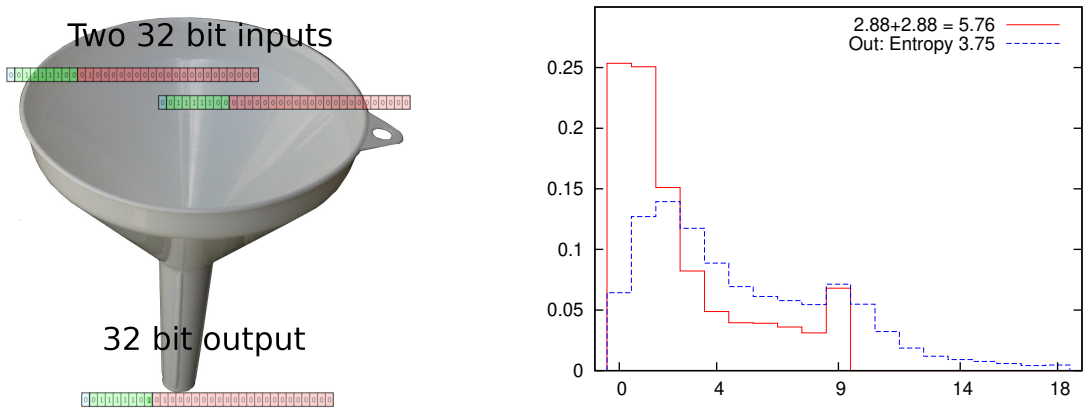


Figure 1. Left: Information Funnel. Computer operators are irreversible. Meaning input state cannot be inferred from outputs. Information is lost. Right: red 0–9 actual distribution of 0–9 digits in C code (Section 4). Dashed blue 0–18 distribution when they are added. Although the output of + is wider and has higher entropy (information content, 3.75 bits), it is smoother and has less entropy than the combined entropy (5.76 bits) of the two inputs to + [8].

4. Failed Disruption Propagation (FDP) in Nested Software

In order to judge the variation in Failed Disruption Propagation (FDP) with depth, we took some highly nested benchmark C code (i.e. VIPS) and inserted mutations into it at random [8]. We measured how deep the mutation was, and followed Voas' PIE framework [27] to measure which mutations were indeed **E**xecuted, which did change (**I**nfect) state and then did that state change **P**ropagate from the mutation's location to impact the outputs. Figure 4 shows in this example, random C code changes deeper than 30 levels of function nesting tend to show FDP, i.e. deeper human written code can be more robust.

VIPS is well established C code which in addition to function's arguments and return values makes heavy used of pointers and shared data. This means information flows both through the nested hierarchy of function calls (like in genetic programming, next section) but also short circuits the hierarchy by passing information via global data. Nevertheless we do see a weak relationship between run time nesting depth and degree of failed disruption propagation. If we exclude mutations which stopped the program immediately, e.g. with a segmentation error, Figure 4 shows there is indeed some variation between mutations with FDP and those without and their execution depth.

The PARSEC suite of benchmarks has been used to test parallel super computers for NASA. It mostly consists of numeric algorithms but also includes some image processing tools, including VIPS [28, 29]. VIPS is an image processing library of 90 000 lines of C source code. We chose the vipsthumbnail application as it was known to contain some highly nested code. vipsthumbnail runs in parallel threads (hence it is non-deterministic). It takes a large image and create a small image (a thumbnail). By default the output is 128 pixels wide. We used a combination of Linux perf and GDB [8] to profile vipsthumbnail and select just the parts of the VIPS library which are heavily used in thumbnail generation (Figure 3). Almost all the unused C code was removed so it could not be mutated This reduced the total library from 90 000 to 7 328 lines of code spread over 37 files. We used the genetic improvement tool Magpie[1] [30] to select uniformly at random (independent of their depth) 1000 locations in the active C code to mutate and record their runtime execution depth and their impact, if any, on the program's output. The results are summarised in Table 1.

Table 1 shows the code is robust to most mutations. Only 16.4% of mutations caused a runtime error. Most of these stopped the program immediately, and there were only 37 where

[1] `https://github.com/bloa/magpie`

Figure 2. Lorenz Butterfly considered harmful. Lorenz [26] viewed the world's weather as being chaotic and posed the question "Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?" but entropy loss makes software robust not chaotic.

Figure 3. Left: 3264×2448 input image. Right: 128×96 thumbnail image generated by VIPS

Table 1. 1000 random Magpie VIPS mutants

Compiled, ran correct output	526	Correct output	438
		Mutation is identical to original code	88
Failed to compile	302		
Failed to run correctly or gave incorrect output	164	exception	127
		output error	37
Magpie TypeError	8		

the program terminated as usual but it generated output that was not identical to the unmutated code. We know these 37 satisfy the PIE frame work. We selected at random 25 of them and looked for 25 mutants amongst the 438 which changed the code and gave the right output. In both cases we select mutants which are executed and which do change the program's internal state and we measure their execution depth. In most cases the mutants are executed more than once, typically thousands of times, and at more than one depth. The 25 mutants which encountered FDP and so gave the right answer (red) and the 25 (of 37) where the mutants disruption reached the output (blue dashed) are plotted in Figure 4. Notice deep mutations ($y > 30$) tend not to impact VIPS thumbnail output.

5. Convergence/Stasis in Genetic Programming (GP)

In a series of experiments we have run genetic programming (GP) on benchmark problems which show GP populations continue to evolve and find better solutions even when run for up to one million generations and evolving trees with up to two billion nodes. Due to a series of innovations [31, 6, 32, 33, 34, 35, 19, 36, 37] in Andy Singleton's GPquick [38] these experiments can be run in days or weeks (rather than years or decades needed by Lenski).

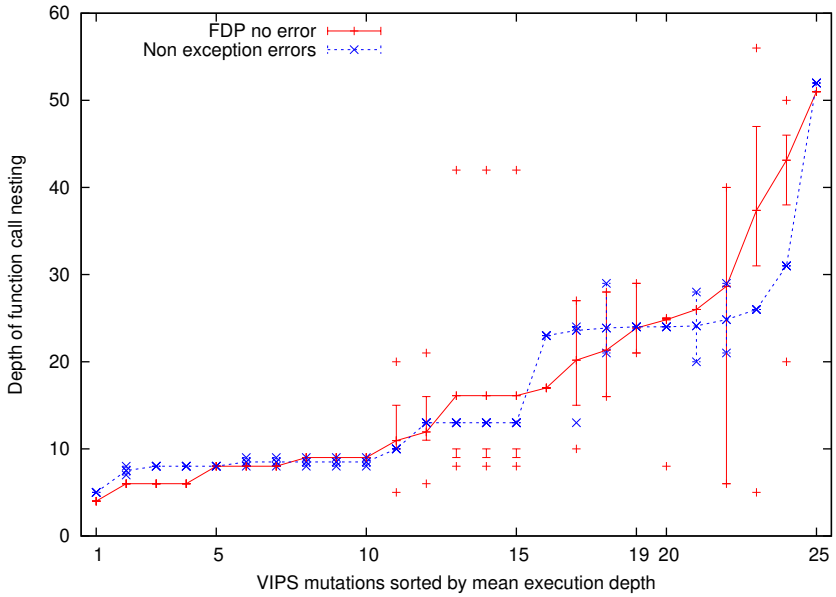


Figure 4. 25 VIPS mutations with no impact (mean depth +) and 25 which change output (mean depth ×). Error bars show interquartile range. +× also show min and max depth. Notice mutations with greater average depth tend not to impact VIPS thumbnail output.

These experiments show that GP runs may find thousands of innovations during an extended run and although fitness improvement continues, it slows (Figure 5). These runs were allowed to produce enormous trees and the rate of improvement is inversely proportional to the size of the trees because failed disruption propagation in these strictly hierarchical systems means only genetic events near the tree's output make any difference (worse or better).

In tree GP programs are usually drawn as inverted trees (e.g. Figures 6 and 7), with the output being taken from the root node, at the top of the diagram and the program's inputs are fed into it via the tree's leafs. The leafs are shown towards the bottom of the diagram.

Figure 6 shows failed disruption propagation where the disruption is introduced at runtime in an integer problem. The shaded nodes indicate the parts of the tree where inserting runtime disruptions has some impact on the program's output. FDP means in the bulk of the tree the disruption has no impact (not shaded). As trees get bigger the sensitive area near the output shrinks as a fraction of the whole tree.

Figure 7 shows an evolved floating point tree where the disruption is to change the program and then re-evaluate it on all the test cases. The coloured oval nodes show the difference between the evaluation in the original tree and the modified tree. The large black oval immediately above the code disruption shows initially the internal evaluation is disrupted on all test cases. As with Figure 1, the system is hierarchical and once information about the disruption is lost it is not recovered higher up the tree. Thus if on a given test case the evaluation between the original (mother) program and her offspring becomes the same, the evaluation above that point (i.e. closer to the output) will remain the same. In particular if FDP occurs for a particular test case then the evaluation of the two programs at the root node will be the same on that test case. So they have the same output. If FDP occurs somewhere on all test cases, then it will succeed in hiding the disruption and both programs will have the same output. Typically this means they have the same fitness scores.

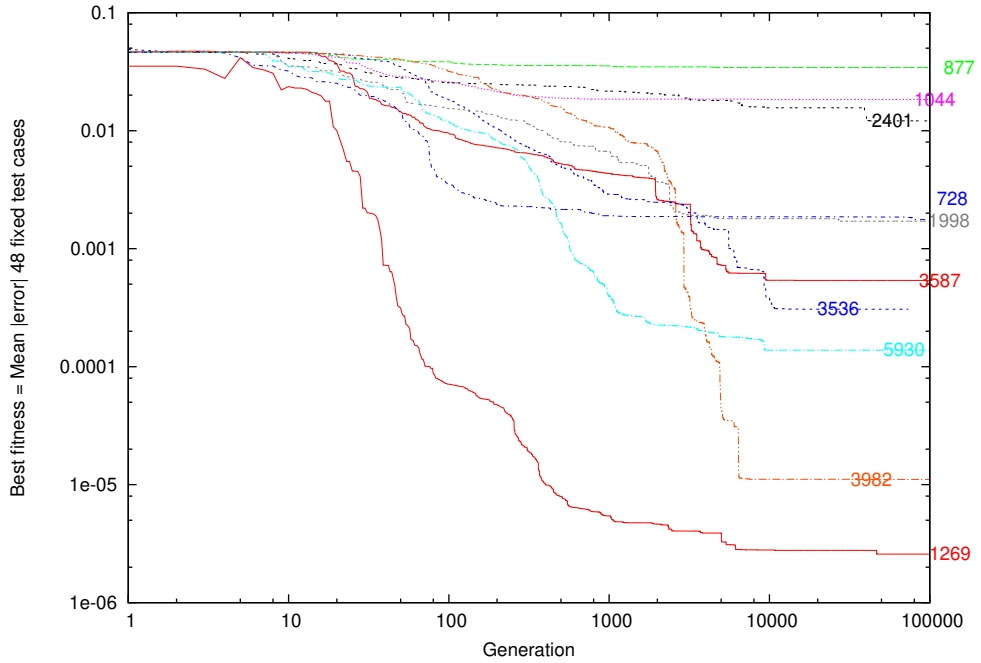


Figure 5. Evolution of mean absolute error in ten GP runs of Sextic polynomial [4] with population of 500. Runs to 100 000 generations (2 stopped early). Labels give number of generations when fitness got better [6].

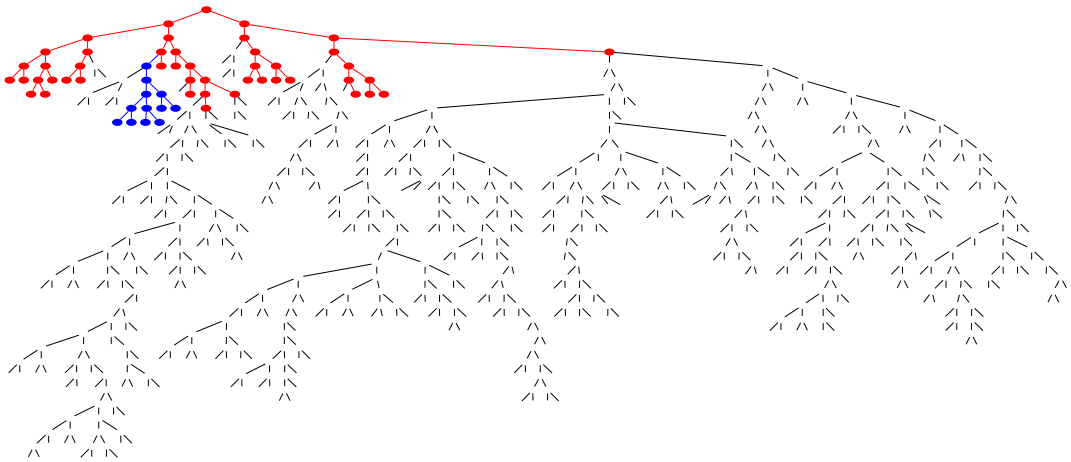


Figure 6. Example evolved solution (size 771) to Fibonacci problem [39]. Coloured shading shows external impact of disrupting internal runtime evaluation by increasing it by 1 (smallest possible change). Disruption at red nodes cause 16-20 test cases to fail. Blue 1 test case fails. As predicted by information theory, failed disruption propagation (FDP) means inserting run time errors far from the output (top) has little effect. Thus almost the whole program is not affected on any test case (white).

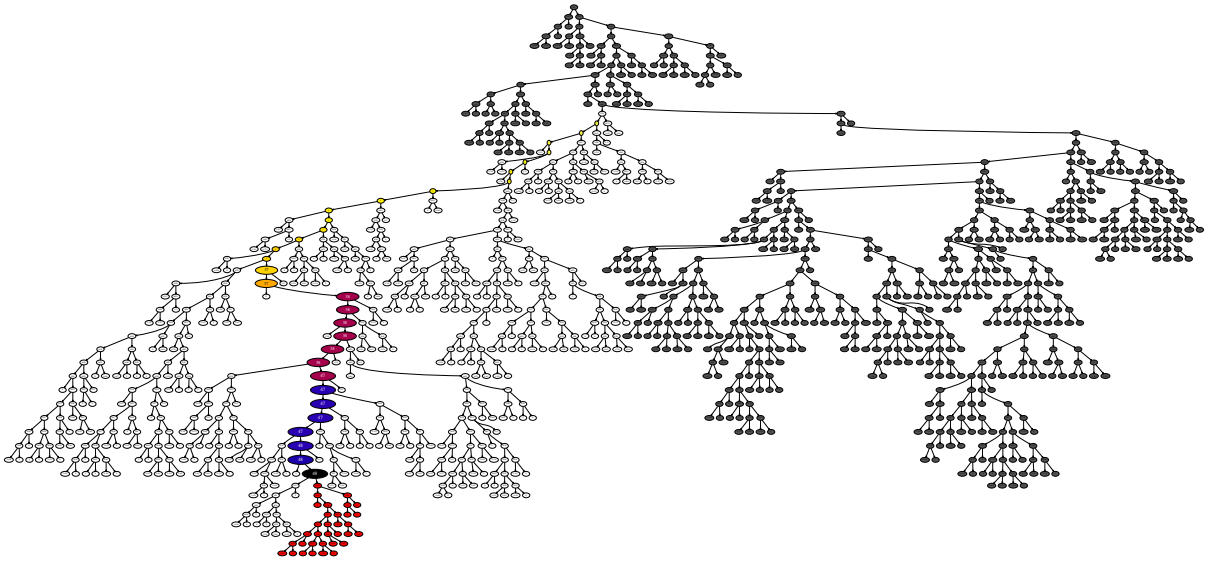


Figure 7. Trace of impact of crossover (red) in evolved tree. Replaced and new code are both evaluated on the test set (48 tests) [33, 40]. The size and number in each node gives the number of test cases where the evaluation of the parent and child are not identical. Their average evaluation difference is indicated on a log scale by the node's colour. Average differences greater than 0.01 are shown with dark colours, less than 0.01 by brighter colours. Brightest yellow shows smallest non-zero difference (RMS 3.1 10^{-10}). If, as here, parent and child evaluations are identical before reaching the root node, the remainder of the evaluation is not needed (gray nodes) [35].

In Figure 7 the number of test cases where evaluation is different in the original and child program is given as a number in the coloured nodes. As expected, this falls monotonically as we move away from the disrupted code towards the root node. Although information theory does not guarantee this (and counter examples do occur), in our floating point trees the size of the difference in evaluation tends to fall. Figure 7 shows this using the colour of the shading, with dark (black or blue) colours indicating large differences and fainter (yellow) showing evaluation in the two programs are almost identical. If, as in Figure 7, at any point evaluation of the parent and child programs become identical, evaluation at their root nodes must be identical, and so their outputs must be identical. Hence evaluation can stop early and we can simply copy the results from the first program to be the outputs of the new program.

Typically there is considerable variation between different trees in different GP runs, nevertheless by taking averages across a large number of disruption locations Figure 8 is able to plot a regression line showing the rapid exponential impact of FDP even when internal evaluation is totally randomised. With floating point, information loss is slower but still rapid and so we again see exponential fall off so that deep disruptions are unlikely to have any impact at all.

6. Implications for Future AI

In nested systems information once lost cannot be recovered. In particular as learning adapts existing programs, information about the adaptation once lost during evaluation of the new program variant can have no impact of the program's outputs. That is, there is no learning signal; externally we cannot tell if the adaptation was good or not. Without feedback, learning becomes random and so continual improvement requires good information flow from the adaptation sites to the program's outputs.

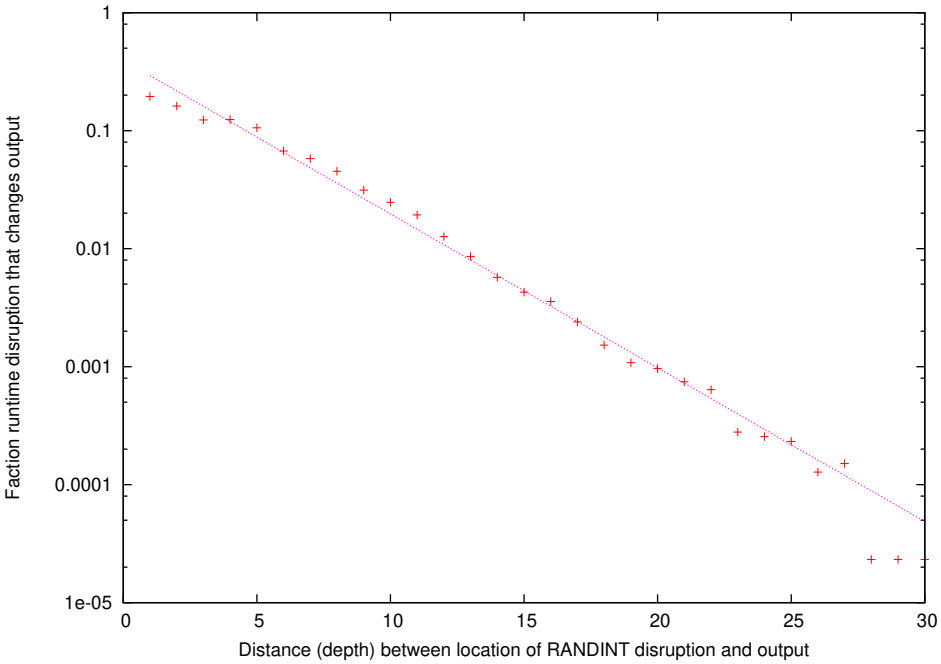


Figure 8. Complete information loss (FDP) becomes almost certain as depth increases when disrupting a large solution to Fibonacci problem with a random value [39]. Dotted line $\approx 0.4\,e^{-0.3}$ shows exponential fall in fraction of run time disruptions reaching output. Meaning the chance of disruption falls by $\frac{1}{2}$ as depth increases by 2.3 levels.

In simple functional systems (previous section) we saw rapid and complete information loss. Similar to AC electrical power flow there is a skin effect: we can make the electrical conductor as thick as we like but once it is wider than the skin depth, increasing the dimensions of the conductor makes very little difference to its ability to carry power. So we can make code as deep as we like, but any change we make will not have impact unless it is near the program's outputs. Unless it has external impact, it will be hard to measure if the change was a learning step in the right direction or not.

In practice in digital computing the "skin depth" can be surprisingly small. In the previous section, for a simple 32 bit integer experiment it was 2.3 function nesting levels. Of course in systems we construct we can bias the location of changes to be near the surface, but then we risk our system growing in complexity and size and so containing large volumes of code which is out of the reach of future changes. Such systems might eventually carry the burden of huge fossilised legacy code. Perhaps it was useful in the past but cannot now be maintained.

We suggest instead of having populations of huge trees, which resemble redwood sequoia trees and are composed almost entirely of inert heartwood, perhaps we should investigate populations which resemble mangroves (left Figure 10). Such systems might be composed of many evolving trees, each being small and shallow but they would be embedded in a data rich sea. Each new tide brings new inputs and moves outputs from one part of the forest to other data processing trees. Figure 10 suggests other naturally occurring porous systems with a high surface area, which might inspire development of thin learning systems, where adaptable code is closely linked to a data rich environment.

Information theory is universal and so unless counter measures are taken we anticipate in future scaling AI systems will have to take into account how to ensure most changes have some measurable impact by ensuring there are short or at least low entropy loss routes from the

site of the change to some measurable point. In evolutionary computing terms, these are the mutation or crossover points and the survival of the individual. If we are to evolve embodied intelligence [41], as with Biology, we may distinguish between mutations etc. to the design (analogous to DNA) and how they lead to changes of form (analogous to the organisms body or phenotype). Again if feedback on the new design is to be available the information chain from design changes to body/mind changes to measurable impact in the environment cannot lose all the entropy. In digital devices such as software, we suggest, even in large systems, the route from genotype (design) change to fitness measurement (phenotype survival) needs to be short.

Large complex systems are not inherently robust but if they are not they cannot be built or survive. If a large system relies on all of its components working but they are fragile. It will never be assembled. As it gets bigger the construction team will spend more and more of its time fixing the components that have just broken, rather than completing construction.

In terms of exactly how we will scale AI the hope remains it can be assembled (or self assemble) from small shallow structures. We can easily foresee a danger that such composites themselves lose entropy and so become robust and hence do not easily adapt. Perhaps the fitness environment (e.g. the sea or atmosphere in Figure 10) will not be sufficient and our evolving architecture will have to include low entropy loss high fidelity channels (analogous to nerve cells or optical fibres) embedded in the digital matrix. However, as always, there is a danger that mandating this means we remain wedded to human designed systems where evolution makes only short term adaptations rather than facilitating long term progress.

7. Artificial Creativity and Uptake of Evolutionary Artificial Intelligence

The creative power of AI in the form of evolutionary computing has been known for several decades [42]. For example, *Creative Evolutionary Systems* [43] contains chapters from computer generated music to novel fighter jet combat [44, p272–276], whilst Koza *et al.*'s 2003 book [45] stresses genetic programming as a routine invention machine. Indeed the annual EvoMusArt conference has been demonstrating computer creativity in the arts since its inception in 2012. Similarly game playing has long been an application of AI, e.g. Checkers (draughts) [46, 47]. The creation of learning based machines which surpass the performance of all humans, has not diminish human interest in games or sports. For example, people still run, even though they know they will never be the best human runner in the world nor outpace a motorcycle. Similarly people play chess and other games despite knowing that superhuman AI performance is readily available. As with art, there are long established conferences and journals dedicated to evolutionary and other learning based approaches to software based game playing, e.g. the IEEE Conference on Games, which started (as CIG) in 2005.

A question raised recently [48, 49] was how much is GP used? Of course most industrialists are not interested in papers. Indeed they may have sound commercial reasons for not publicising their results or even saying what they are working on. This means numbers based on published work will always be an underestimate [50]. Nevertheless data sampled from the genetic programming bibliography for last year (2023) suggests $38\pm5\%$ of published papers are primarily on applications which just happen to use GP (Figure 9). Many applications relate to health (e.g. [51, 52, 53, 54, 55, 56, 57, 58, 59]), civil engineering (e.g. [60, 61, 62, 63, 64, 65, 66, 67, 68]) or solid state materials, e.g. batteries [69, 70, 71, 72, 73].

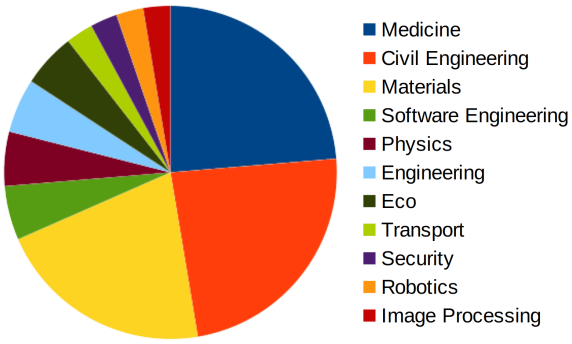
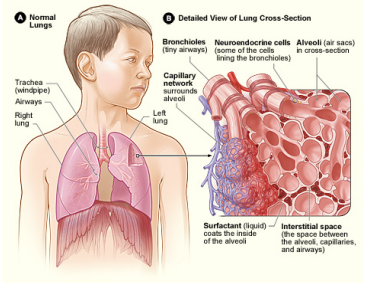
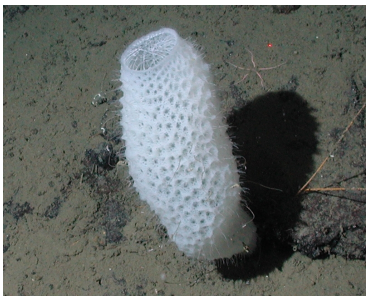


Figure 9. An estimated 38±5% of papers from 2023 in the GP bibliography are on applications

Mangrove a



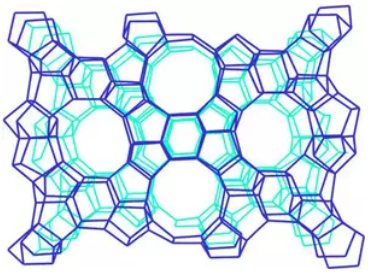
Lungs b [74]



Sponge c

Coral d

Pumice e



Zeolite f [75]

Figure 10. Possible natural sources of inspiration for software architectures suited to prolonged evolution. They have a large surface area so that their volume lies close to a surface. Such structures could provide a large space for evolving software whilst keeping most of the code near to its environment (Wikimedia a b c d e, [75]f)

8. Conclusions

Information theory predicts that code changes will suffer failed disruption propagation FDP, which makes both deeply nested automatically generated code evolved by genetic programming (GP) and human written software resistant to change. In GP with an integer representation the impact of even very large run time disruptions on average halved for each increase in depth of 2.3 levels.

Therefore we advocate research into porous high surface area shallow code "mangrove" architectures, where the adapting code is closely connected to a data rich environment. To ensure that updates continue to have impact, the system must have limited robustness and so limited code depth. Figure 10 shows a few natural structures with high surface areas, which might inspire continual adaptation EI architectures.

References

[1] Lenski R E *et al.* 2015 *Proceedings of the Royal Society B* **282** ISSN 0962-8452 URL http://dx.doi.org/10.1098/rspb.2015.2292

[2] Good B H *et al.* 2017 *Nature* **551** 45–50 URL http://dx.doi.org/10.1038/nature24287

[3] Wang R J *et al.* 2023 *Science Advances* **9** eabm7047 URL http://dx.doi.org/10.1126/sciadv.abm7047

[4] Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA, USA: MIT Press) ISBN 0-262-11170-5 URL http://mitpress.mit.edu/books/genetic-programming

[5] Poli R *et al.* 2008 *A field guide to genetic programming* (Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk) (With contributions by J. R. Koza) URL http://www.gp-field-guide.org.uk

[6] Langdon W B and Banzhaf W 2022 *Artificial Life* **28** 173–204 ISSN 1064-5462 invited submission to Artificial Life Journal special issue of the ALIFE'19 conference URL http://dx.doi.org/10.1162/artl_a_00360

[7] Langdon W B 2022 *Genetic Programming and Evolvable Machines* **23** 71–104 ISSN 1389-2576 URL http://dx.doi.org/10.1007/s10710-021-09405-9

[8] Langdon W B and Clark D 2024 *13th International Workshop on Genetic Improvement @ICSE 2024* ed An G *et al.* (Lisbon: ACM) pp 1–8 best paper URL http://dx.doi.org/10.1145/3643692.3648259

[9] Petke J *et al.* 2021 *ESEC/FSE 2021, Ideas, Visions and Reflections* ed Avgeriou P and Zhang D (Athens, Greece: ACM) pp 1475–1478 URL http://dx.doi.org/10.1145/3468264.3473133

[10] Harman M and Jones B F 2001 *Information and Software Technology* **43** 833–839 ISSN 0950-5849 URL http://dx.doi.org/10.1016/S0950-5849(01)00189-6

[11] Jahangirova G *et al.* 2016 *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA'16)* (Saarbruecken, Germany: ACM) pp 247–258 URL http://dx.doi.org/10.1145/2931037.2931062

[12] Beyer H G 2001 *The Theory of Evolution Strategies* Natural Computing Series (Springer) ISBN 3-540-67297-4

[13] Reeves C R and Rowe J E 2003 *Genetic Algorithms–Principles and Perspectives: A Guide to GA Theory* (Kluwer Academic Publishers)

[14] Li M and Vitanyi P M B 1992 *Journal of Computer and System Sciences* **44** 343–384

[15] Cilibrasi R *et al.* 2004 *Computer Music Journal* **28** 49–67 URL http://homepages.cwi.nl/~paulv/papers/music.pdf

[16] Cilibrasi R and Vitanyi P M B 2005 Automatic meaning discovery using Google v2 URL http://www.arxiv.org/abs/cs.CL/0412098

[17] Cilibrasi R L and Vitanyi P M B 2007 *IEEE Transactions on Knowledge and Data Engineering* **19** 370–383 ISSN 1041-4347 URL http://dx.doi.org/10.1109/TKDE.2007.48

[18] Langdon W B and Poli R 1997 *Soft Computing in Engineering Design and Manufacturing* ed Chawdhry P K *et al.* (Springer-Verlag London) pp 13–22 ISBN 3-540-76214-0 URL http://dx.doi.org/10.1007/978-1-4471-0427-8_2

[19] Langdon W B 2022 A trillion genetic programming instructions per second ArXiv URL https://arxiv.org/abs/2205.03251

[20] Silva S *et al.* 2012 *Genetic Programming and Evolvable Machines* **13** 197–238 ISSN 1389-2576 URL http://dx.doi.org/10.1007/s10710-011-9150-5

[21] Poli R and McPhee N F 2013 *Theory and Principled Methods for the Design of Metaheuristics* Natural Computing Series ed Borenstein Y and Moraglio A (Springer) pp 181–204 URL http://dx.doi.org/10.1007/978-3-642-33206-7_9

[22] Card S W 2010 *GECCO 2010* ed Card S W and Borenstein Y (Portland, Oregon, USA: ACM) pp 1851–1854 URL http://dx.doi.org/10.1145/1830761.1830815

[23] Hulme D *et al.* 2024 Solving machine consciousness: Theory and approach Conscium

[24] DeMillo R A *et al.* 1978 *IEEE Computer* **11** 31–41 URL http://dx.doi.org/10.1109/C-M.1978.218136

[25] Langdon W B and Petke J 2015 *Complex Systems Digital Campus E-conference, CS-DC'15* Proceedings in Complexity ed Parrend P *et al.* (Springer) pp 203–211 invited talk URL http://dx.doi.org/10.1007/978-3-319-45901-1_24

[26] Lorenz E N 1993 *The Essence of Chaos* Jessie and John Danz lectures (Seattle, USA: University of Washington Press) chap Appendix 1, pp 181–184 URL https://uwapress.uw.edu/book/9780295975146/the-essence-of-chaos/

[27] Voas J M and Miller K W 1995 *IEEE Software* **12** 17–28 ISSN 0740-7459 URL http://dx.doi.org/10.1109/52.382180

[28] Martinez K and Cupitt J 2005 *Proceedings of the 2005 International Conference on Image Processing, ICIP* (Genoa, Italy: IEEE) pp 574–577 URL http://dx.doi.org/10.1109/ICIP.2005.1530120

[29] Langdon W B and Clark D 2024 *EuroGP 2024: Proceedings of the 27th European Conference on Genetic*

Programming (*LNCS* vol 14631) ed Giacobini M *et al.* (Aberystwyth: Springer Verlag) pp 209–226 URL http://dx.doi.org/10.1007/978-3-031-56957-9_13
[30] Blot A and Petke J 2022 MAGPIE: Machine automated general performance improvement via evolution of software arXiv 2208.02811 URL http://dx.doi.org/10.48550/arxiv.2208.02811
[31] Langdon W B 2019 *GECCO '19 Companion* ed Doerr C (Prague, Czech Republic: ACM) pp 63–64 URL http://dx.doi.org/10.1145/3319619.3326770
[32] Langdon W B 2020 Multi-threaded memory efficient crossover in C++ for generational genetic programming ArXiv 2009.10460 URL http://arxiv.org/abs/2009.10460
[33] Langdon W B 2021 *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming* (*LNCS* vol 12691) ed Hu T *et al.* (Virtual Event: Springer Verlag) pp 229–246 URL http://dx.doi.org/10.1007/978-3-030-72812-0_15
[34] Langdon W B 2021 *Proceedings of the Genetic and Evolutionary Computation Conference Companion* GECCO '21 ed Chicano F *et al.* (Internet: Association for Computing Machinery) pp 253–254 URL http://dx.doi.org/10.1145/3449726.3459437
[35] Langdon W B 2021 *Genetic Programming Theory and Practice XVIII* Genetic and Evolutionary Computation ed Banzhaf W *et al.* (East Lansing, MI, USA: Springer) pp 143–164 URL http://dx.doi.org/10.1007/978-981-16-8113-4_8
[36] Langdon W B 2022 *ACM Transactions on Evolutionary Learning and Optimization* **2** ISSN 2688-299X URL http://dx.doi.org/10.1145/3539738
[37] Langdon W B 2022 *Complex Systems* **31** 287–309 ISSN 0891-2513 URL http://dx.doi.org/10.25088/ComplexSystems.31.3.287
[38] Singleton A 1994 *BYTE* 171–176 ISSN 0360-5280 URL http://www.assembla.com/wiki/show/andysgp/GPQuick_Article
[39] Langdon W B 2022 *Proceedings of the Genetic and Evolutionary Computation Conference Companion* GECCO '22 ed Trautmann H *et al.* (Boston, USA: Association for Computing Machinery) pp 574–577 URL http://dx.doi.org/10.1145/3520304.3528878
[40] Langdon W B *et al.* 2021 *5th Workshop on Landscape-Aware Heuristic Search* GECCO 2021 Companion ed Veerapen N *et al.* (Internet: ACM) pp 1683–1691 URL http://dx.doi.org/10.1145/3449726.3463147
[41] Hughes J *et al.* 2022 *IOP Conference Series: Materials Science and Engineering* **1261** 012001 URL http://dx.doi.org/10.1088/1757-899X/1261/1/012001
[42] Langdon W B 2024 *Communications of the ACM* **67** 8 ISSN 0001-0782 letter to the editor URL http://dx.doi.org/10.1145/3654698
[43] Bentley P J and Corne D W 2001 *Creative Evolutionary Systems* ed Bentley P J and Corne D W (Morgan Kaufmann) pp 1–75 ISBN 1-55860-673-4 URL http://dx.doi.org/10.1016/B978-155860673-9/50035-5
[44] Smith R E 2019 *Rage Inside the Machine–the prejudice of algorithms, and how to stop the internet making bigots of us all* (Bloomsbury business) ISBN 9781472963888
[45] Koza J R *et al.* 2003 *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Kluwer Academic Publishers) ISBN 1-4020-7446-8 URL http://dx.doi.org/10.1007/0-387-26417-5_1
[46] Samuel A L 1959 *IBM Journal* **3** 210–229
[47] Fogel D B 2001 *Blondie24: Playing at the Edge of AI* (Morgan Kaufmann) ISBN 1-55860-783-8 URL https://en.wikipedia.org/wiki/Blondie24
[48] Langdon W B 2023 *Genetic Programming and Evolvable Machines* **24** Article number: 19 ISSN 1389-2576 special Issue: Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection URL http://dx.doi.org/10.1007/s10710-023-09467-x
[49] Bartoli A *et al.* 2023 *Genetic Programming and Evolvable Machines* **24** Article number: 23 ISSN 1389-2576 special Issue: Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection URL http://dx.doi.org/10.1007/s10710-023-09471-1
[50] Langdon W B 2023 *Genetic Programming and Evolvable Machines* **24** Article number: 26 ISSN 1389-2576 special Issue: Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection URL http://dx.doi.org/10.1007/s10710-023-09474-y
[51] Nguyen S *et al.* 2023 *International Journal of Disaster Risk Reduction* **97** 104004 ISSN 2212-4209 URL http://dx.doi.org/10.1016/j.ijdrr.2023.104004
[52] Andelic N and Baressi Segota S 2023 *Cancers* **15** article no. 3411 ISSN 2072-6694 URL http://dx.doi.org/10.3390/cancers15133411
[53] Guidetti V *et al.* 2023 *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)* URL http://dx.doi.org/10.1109/DSAA60987.2023.10302622
[54] Hurta M *et al.* 2023 *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* pp 3782–3787 ISSN 2156-1133 URL http://dx.doi.org/10.1109/BIBM58861.2023.10385615
[55] Bartlett L K *et al.* 2023 *Proceedings of the 45th Annual Meeting of the Cognitive Science Society* ed Goldwater

M *et al.* (Sydney, Australia) pp 2833–2839 URL http://hdl.handle.net/2299/27181
[56] Romano J D *et al.* 2023 *Computational Toxicology* **25** 100261 ISSN 2468-1113 URL http://dx.doi.org/10.1016/j.comtox.2023.100261
[57] Javed N *et al.* 2023 *AISB 2023 convention proceedings. The Society for the Study of Artificial Intelligence and Simulation Behaviour* ed Mueller B (Swansea, UK) pp 43–50 URL http://eprints.lse.ac.uk/id/eprint/118805
[58] MacLachlan J *et al.* 2023 *Proceedings of the 2023 Genetic and Evolutionary Computation Conference* GECCO '23 ed Silva S *et al.* (Lisbon, Portugal: Association for Computing Machinery) pp 1409–1417 silver 2023 HUMIES URL http://dx.doi.org/10.1145/3583131.3590434
[59] Hurta M *et al.* 2023 *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)* pp 155–160 ISSN 2473-2117 URL http://dx.doi.org/10.1109/DDECS57882.2023.10139399
[60] Degefa A B *et al.* 2023 *Sustainability* **15** Article No. 15471 ISSN 2071-1050 URL http://dx.doi.org/10.3390/su152115471
[61] Ozbayrak A *et al.* 2023 *Arabian Journal for Science and Engineering* **48** 5347–5370 ISSN 2193-567X URL http://dx.doi.org/10.1007/s13369-022-07445-6
[62] Martin-Alcantara A *et al.* 2023 *Sustainable Energy Technologies and Assessments* **56** 103053 ISSN 2213-1388 URL http://dx.doi.org/10.1016/j.seta.2023.103053
[63] Yin Z *et al.* 2023 *Ocean Engineering* **285** 115372 ISSN 0029-8018 URL http://dx.doi.org/10.1016/j.oceaneng.2023.115372
[64] Ismail M K *et al.* 2023 *Engineering Structures* **295** 116806 ISSN 0141-0296 URL http://dx.doi.org/10.1016/j.engstruct.2023.116806
[65] Althoey F *et al.* 2023 *Case Studies in Construction Materials* **18** e01774 ISSN 2214-5095 URL http://dx.doi.org/10.1016/j.cscm.2022.e01774
[66] Sadat Hosseini A *et al.* 2023 *Ocean Engineering* **279** 114465 ISSN 0029-8018 URL http://dx.doi.org/10.1016/j.oceaneng.2023.114465
[67] Al-Aghbari M and M Gujarathi A 2023 *Geoenergy Science and Engineering* **228** 211967 ISSN 2949-8910 URL http://dx.doi.org/10.1016/j.geoen.2023.211967
[68] Xue X *et al.* 2023 *Alexandria Engineering Journal* **81** 599–619 ISSN 1110-0168 URL http://dx.doi.org/10.1016/j.aej.2023.09.053
[69] Di Capua G *et al.* 2023 *26th International Conference, EvoApplications 2023* (*LNCS* vol 13989) ed Correia J *et al.* (Brno, Czech Republic: Springer Verlag) pp 461–474 URL http://dx.doi.org/10.1007/978-3-031-30229-9_30
[70] Di Capua G *et al.* 2023 *2023 IEEE International Symposium on Circuits and Systems (ISCAS)* ISSN 2158-1525 URL http://dx.doi.org/10.1109/ISCAS46773.2023.10181456
[71] Vandana *et al.* 2023 *2023 IEEE 3rd International Conference on Sustainable Energy and Future Electric Transportation (SEFET)* URL http://dx.doi.org/10.1109/SeFeT57834.2023.10244776
[72] Milano F *et al.* 2023 *2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)* pp 35–40 URL http://dx.doi.org/10.1109/MetroAutomotive57488.2023.10219104
[73] Hosseinhashemi S *et al.* 2023 *Journal of Energy Storage* **73** 109046 ISSN 2352-152X URL http://dx.doi.org/10.1016/j.est.2023.109046
[74] Langdon W B 2022 *Proceedings of 2022 International Conference on Embodied Intelligence, EI-2022* (*IOP Conference Series: Materials Science and Engineering* vol 1292) ed Iida F *et al.* (Internet, Cambridge: IOP Publishing) p 012021 URL http://dx.doi.org/10.1088/1757-899X/1292/1/012021
[75] Castro M *et al.* 2016 *Chem. Eur. J.* **22** 15307–15319 URL http://dx.doi.org/10.1002/chem.201600511