Language Model Crossover: Variation through Few-Shot Prompting

Elliot Meyerson
Cognizant AI Labs
elliot.meyerson@cognizant.com

Mark J. Nelson
American University
mnelson@american.edu

Herbie Bradley
University of Cambridge & CarperAI
hb574@cam.ac.uk

Arash Moradi
New Jersey Institute of Technology
am3493@njit.edu

Amy K. Hoover
New Jersey Institute of Technology
ahoover@njit.edu

Joel Lehman
CarperAI
joel@stability.ai

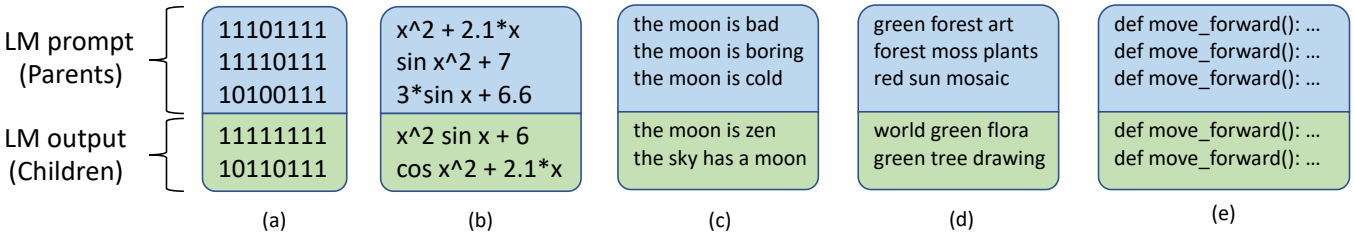


Figure 1: *Language Model Crossover (LMX).* New candidate solutions are generated by concatenating parents into a prompt, feeding the prompt through a large pre-trained language model (LM), and collecting offspring from the output. The enormity and breadth of the dataset on which the LM was trained, along with its ability to perform in-context learning, enables LMX to generate high-quality offspring across a broad range of domains. Domains demonstrated in this paper include (a) binary strings, (b) mathematical expressions, (c) English sentences, (d) image generation prompts, and (e) Python code; many more are possible. When integrated into an optimization loop, LMX serves as a general and effective engine of text-representation evolution.

ABSTRACT

This paper pursues the insight that language models naturally enable an intelligent variation operator similar in spirit to evolutionary crossover. In particular, language models of sufficient scale demonstrate in-context learning, i.e. they can learn from associations between a small number of input patterns to generate outputs incorporating such associations (also called few-shot prompting). This ability can be leveraged to form a simple but powerful variation operator, i.e. to prompt a language model with a few text-based genotypes (such as code, plain-text sentences, or equations), and to parse its corresponding output as those genotypes' offspring. The promise of such language model crossover (which is simple to implement and can leverage many different open-source language models) is that it enables a simple mechanism to evolve semantically-rich text representations (with few domain-specific tweaks), and naturally benefits from current progress in language models. Experiments in this paper highlight the versatility of language-model crossover, through evolving binary bit-strings, sentences, equations, text-to-image prompts, and Python code. The conclusion is that language model crossover is a promising method for evolving genomes representable as text.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

neuroevolution, recombination, language models

1 INTRODUCTION

Large language models (LMs; [6, 9]) have achieved impressive results in many natural language domains, such as question-answering [14, 23, 49], code-generation [12, 48], and few-shot classification [9, 68]. One popular type of LM is trained on corpora of human-authored text to predict the next token from previous ones, i.e. auto-regressive LMs (e.g. GPT-3), which at their core model a distribution of likely output sequences given an input sequence or *prompt*. In zero-shot learning, an LM generates an output response from a single input sequence. However, another popular prompting paradigm is *few-shot prompting* [9], wherein the input to a LM is a few examples of desired input-output behavior (e.g. how to classify a sentence's sentiment) preceding a new target input that the model is to classify. To some extent such LMs can *meta-learn* from a few natural-language examples how to perform a desired task [11, 83].

One reason this ability is exciting because it highlights how LMs can in effect be seen as very powerful pattern-completion engines. Few-shot prompting works because the LM can "guess the pattern" behind a few input/output pairs and generalize its behavior to a new target input (provided at the end of the few-shot prompt). The central insight of this paper is is that, interestingly, the pattern-completion ability of few-shot prompting can be leveraged to create a form of intelligent evolutionary crossover.

For example, if three text-based genotypes are drawn from a population and concatenated into a prompt, an ideal pattern-completion engine would analyze their commonalities and generate a new (fourth) genotype that qualitatively follows from the same distribution. In effect such an operator would combine aspects of the input genotypes, and indeed, an experiment in section 4.1 demonstrates empirically that LMs enable this with binary strings. Theoretically we also connect this form of *LM crossover* (LMX) to estimation of distribution algorithms (EDAs; [2, 44]), wherein LMX can be seen as building an implicit probabilistic model of the input parent genotypes from which to sample a new offspring, *through a single forward pass of the LM*. From the perspective of pattern-completion, this operator should naturally improve as LMs increase in capabilities (which experiments here validate); furthermore, to increase performance the method can easily leverage the rise of open-source domain-specific LMs that match a target domain (e.g. LMs that focus on code, when the target domain is to evolve code), often with changing only a single line of code to rely on a different hosted model (e.g. through the HuggingFace model repository [90]).

The benefit of LMX is that evolution can easily be conducted in the semantic representation of text, without having to design specific domain-specific variation operators. LMX's versatility is highlighted in experiments with binary strings, style transfer of plain-text sentences, symbolic regression of mathematical expressions, generating images through prompts for a text-to-image model, and generating Python code. The results highlight the potential of the method to produce quality results across domains, often by leveraging the broad ecosystem of pretrained models that can be easily combined in many ways to quantify fitness or diversity, or to cross modalities (i.e. from text to image). Interestingly, LMX may also synergize with recent LM-based mutation techniques [46], and is amenable to similar possibilities such as fine-tuning an LM as a way of accelerating search, although we leave these possibilities for future work.

In short, the main contributions of this paper are to introduce LMX, explore its basic properties, and highlight its versatility through testing it in a variety of domains. We will release an implementation of LMX and code to recreate the main experiments of the paper.

2 BACKGROUND

2.1 Foundation Models

A recent paradigm in ML is to train increasingly large models on internet-scale data, e.g. BERT and GPT-3 on text [9, 22], or DALL-E and stable diffusion on captioned images [63, 65]. Such models are sometimes called foundation models [6], as they provide a broad foundation from which they can be specialized to many specific domains (e.g. with supervised fine-tuning or prompt-engineering). Interestingly, such foundation models have enabled a large ecosystem of specialized models [84] that can be combined in a plug-and-play way (e.g. models that measure sentiment of text [10], summarize text [76], write code [56], rank the aesthetics of images [20, 40, 72], and create high-dimensional embeddings of text or images [64, 92]. One contribution of this paper is to demonstrate how evolutionary methods can easily leverage this growing eco-system to evolve high-quality artifacts in diverse applications.

One particularly exciting class of foundation models are pretrained language models (LMs) that model the distribution of text. While early LMs used markov chains [74] or recurrent neural networks [25], more recently the transformer architecture [82] has enabled significant progress in NLP. The method in this paper focuses on one emergent capability of large transformer-based LMs, i.e. the potential to meta-learn from text examples provided as input to the model when generating an output, which is called in-context learning or few-shot prompting [9, 83]. Importantly, performance at in-context learning improves with model scale [11, 88], implying that methods relying upon this capability will benefit from continuing progress in LM training. This paper highlights how the in-context learning capabilities of autoregressive LMs (such as the popular GPT architecture) naturally enable an intelligent recombination operator. The next section reviews existing methods for intelligent variation in EC.

2.2 Intelligent Variation Operators

Populations in evolutionary algorithms (EAs) generally evolve through high performing candidates solutions being mutated or recombined to form the next generation. Such variation is critical as a primary driver of both exploration and exploitation of the search space [17]. Traditional mutation and recombination operators (such as one-point crossover or bit-flip mutation) do not explicitly seek to model and exploit regularities among high-fitness individuals (or do so in an implicit way [31]), which can cause EAs to be relatively sample-inefficient in some situations when compared to statistical methods [80].

To address this limitation, strategies for generating intelligent variation have been a focus of much research in EC. For example, evolving within a latent space of an ML model [24, 71], or through training models to mimic mutations [37, 46]. One particularly popular such strategy is to build probabilistic models of high-performing individuals or to model elements of the search path taken across recent generations. For example, estimation of distribution algorithms (EDA;[2, 44]), covariance matrix adaptation evolution strategy (CMA-ES; [28]), and natural evolution strategies (NES; [89]) build and sample candidate solutions from an explicit probability distribution. While EDAs estimate the distribution of the solutions that have been sampled, CMA-ES additionally estimates the steps of the search direction. The LMX operator in this paper can be seen similarly as building a probabilistic model of individuals (here of parents, rather than the whole population), and doing so implicitly in the forward-pass of the LM (through in-context learning).

One recent exciting direction for generating variation is to leverage the semantic knowledge of pretrained LMs. This work builds on previous results that highlight the potential of LMs to generate [69] or augment [26] data, although here the focus is on enabling continual evolution. It also is closely related to work demonstrating that LMs can be trained to embody intelligent mutation operators for code [46], i.e. by training a model on changes to code files gathered from GitHub; Lehman et al. [46] also proposed a way to generate domain-specific mutations from hand-designed prompts for LMs. Such operators may require fine-tuning a model on mutation-like training data gathered from GitHub or hand-specifying example mutations, and have been applied only to the domain of code. Instead,

the mechanism exploited here focuses on recombination rather than mutation, is domain-independent (as shown by the diversity of domains targeted by this paper), and is easily implemented with open-source LMs (we experiment with several).

3 APPROACH: LANGUAGE MODEL CROSSOVER (LMX)

The approach in this paper builds from the insight that the objective function used to train many self-supervised LMs, i.e. next-token prediction [9], naturally lends itself to creating an evolutionary variation operator, from which evolutionary algorithms that represent genomes as text can be derived. The reason is that such an objective entails anticipating what comes next from some limited input context, and if that input consists of a few example genotypes, then the ideal anticipation is to continue that pattern, i.e. through suggesting a new genotype from the distribution implied by those examples. In other words, LMs trained by next-token prediction can be seen as learning to become general pattern-completion engines. From this lens, as higher-performing LMs (i.e. those with lower prediction loss on a held-out set) are continually developed, their performance as engines of evolutionary variation should continue to improve. Supporting this idea, when trained over a large amount of diverse examples, LMs demonstrate an increasing capability for in-context learning (i.e. inferring novel associations within the input given at test-time when generating completions) [9, 11, 88].

What is intriguing about this insight is that the variation operator it suggests is (1) simple to implement (i.e. concatenate a few text-based genotypes into a prompt, run it through a LM, and extract a new genotype from its output; we release code implementing it accompanying this paper), (2) relatively domain-independent (i.e. in theory it should be capable of generating meaningful variation for any text representation that has moderate support in the training set, which often encompasses a crawl of the internet), and (3) should suggest increasingly semantically-sophisticated variation with more capable LMs (i.e. reduced test-loss of LMs should correlate with the ability to exploit more subtle input patterns). The experiments that follow add supporting evidence to these claims.

Figure 1 shows from a high level how LMX enables creating a domain-independent evolutionary algorithm for text representations. The basic idea is that given a set of a few text-based genotypes (or bootstrapping from a single genotype using prompt-based mutation [46]), an initial population can be generated through LMX. Then, a standard evolutionary loop can be instantiated by repeated selection and generation of new variation through LMX. In the experiments that follow, we use simple genetic algorithms (GAs; although one experiment instantiates a simple quality diversity algorithm). In theory, however, LMX can be generically applied to most EAs, e.g. multi-objective EAs [15, 18], evolutionary strategies [1, 3], or in support of open-ended evolution [85]. How or if LMX can be applied to EAs that explicitly leverage probabilistic models of genotypes (e.g. EDAs [2, 44], natural evolution strategies [89], or CMA-ES [27, 28]) is an interesting question for future research, although LMX does bear a strong theoretical relationship to EDA algorithms in particular, explored next.

3.1 Connection to EDAs

An EDA constructs an *explicit* probabilistic distribution D fit to the parent set $\{x_1, \ldots, x_M\}$, and samples child solutions x from D [30, 45]. In contrast, a standard GA generates children by sampling from an *implicit* conditional probability distribution $p_g(x \mid [x_1, \ldots, x_M])$ induced by the process of randomly sampling parents and applying a stochastic reproduction operator g (e.g., a crossover operator). LMX occupies an intermediate level of explicitness: The conditional distribution induced by feeding the parent prompt into the LM is *explicit* in that it yields a series of probability distributions over tokens, but is *implicit* in the sense that the internal workings of the distribution are encoded opaquely within the millions or billions of parameters and activations of the LM for a given prompt.

Whatever the level of explicitness, the reason LMX can be viewed as an EDA is that it be seen as constructing a distribution D of parents, from which children are sampled. The key design feature of an EDA is the class of distributions $\mathcal{D}$ to which D belongs. This class $\mathcal{D}$ can range from simple univariate distributions [2, 29] to more complex models like Bayesian networks [59, 60].

What is the class $\mathcal{D}_{\text{LM}}$ from which LMX constructs parent distributions? Due to its in-context learning capabilities [66, 91], the LM can be seen as attempting to infer the generating distribution of the prompt, and to generate continuations accordingly. By concatenating parents in a random order, the implicit signal to the LM is that the list is unordered (i.e. there is little to be inferred from most arbitrary randomly-ordered patterns). These objects must have been sampled from some distribution D, and thus the LM's optimal move is to keep sampling objects from D as it generates output. In other words D_{LM} consists of distributions of *objects that are found in sets that might appear in the universe* of data from which the dataset used to train the LM was drawn. An ideal EDA would select the most probable $D = D_{\text{EDA}} \in \mathcal{D}_{\text{LM}}$ based on the parent set $\{x_1, \ldots, x_M\}$. E.g.,

$$D_{\text{EDA}} = \underset{D \in \mathcal{D}_{\text{LM}}}{\operatorname{argmax}}\, p(D) \prod_{i=1}^{M} p(x_i \mid D), \tag{1}$$

where $p(D)$ is the prior probability of D in $\mathcal{D}_{\text{LM}}$. As the LM becomes a better and better in-context learner, it becomes better able to detect subtler patterns within a prompt of randomly-ordered concatenated parents, and thus

$$p_{\text{LMX}}(x \mid [x_1, \ldots, x_M]) \approx p(x \mid D_{\text{EDA}}). \tag{2}$$

Note that the left side depends on an ordered list of parents, while the right side has removed this dependency on order. This approximation becomes tight as the LM approaches perfect in-context learning, at which point LMX can be viewed exactly as an EDA, i.e.,

$$\text{LMX}([x_1, \ldots, x_M]) \sim D_{\text{EDA}}. \tag{3}$$

Although a faithful application of an EDA may include the full parent population in each parent prompt, the experiments in this paper save compute by sampling a only a small number of parents. Not only are EDAs an efficient type of EA [59, 60], but by comparing LMX to EDAs it may be possible to analyze the optimization behavior of LMX [41] (e.g., global convergence analysis [93]). Overall, the connection to EDAs may help to explain why LMX is effective as an off-the-shelf genetic operator across a wide range of domains;

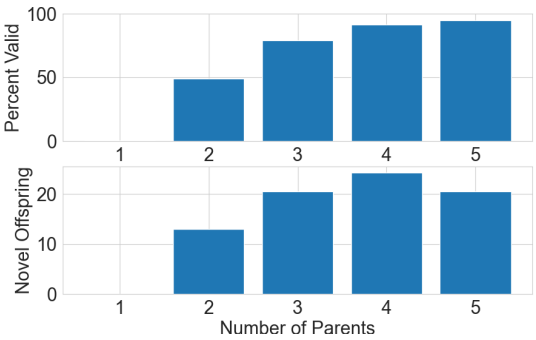


Figure 2: *The effect on LMX from varying the number of parents.* As the number of parent genotypes input into the LM is increased, the percent of valid offspring approaches 100%. The number of novel genotypes generated on average from 20 applications of LMX to a random set of parents reaches its maximum at four parents (while five parents tends to more often produce offspring that duplicate one of the parents exactly). The conclusion is that LMX effectively generates variation from as few as three input genotypes.

a further interesting theoretical property of LMX is its universality (described in appendix A).

4 EXPERIMENTS

This section demonstrates the application of LMX to five domains. Source code will be made available for each domain.

4.1 Illustrative Example: Binary Strings

As an instructive example to explore the properties of LMX, in this section this operator is applied to generate variation in the space of binary strings (e.g. composed of text strings such as "011000").

A first question is whether a pretrained LM (here an 800-million parameter Pythia model [4]), given only a few examples of such genomes, can generate meaningful variation (i.e. without any hard-coded knowledge about the representation). To explore this question, a prompt is generated by concatenating randomly chosen length-6 binary strings separated by newlines; the LM's response (truncated after three new lines) is interpreted as three offspring individuals. Figure 2 shows how often such a prompt will generate valid individuals (i.e. strings of length six composed of 1s and 0s) as a function of number of examples in the prompt, and how many novel offspring (i.e. the size of the set of individuals generated that are distinct from the parents) are generated on average from 20 trials of LMX crossover on the same set of parents (averaged across 20 randomly-sampled parent sets). A follow-up experiment, with length-9 binary strings, demonstrates how LMX in this domain improves with larger LMs (details in appendix B.1). The conclusion is that indeed, LMX can reliably generate novel, valid offspring (from as few as three examples).

A second question is whether LMX can create *heritable* variation. Evolution requires there to be meaningful information transmitted from parents to offspring. One way to explore this is to measure whether a prompt composed of highly-related binary strings produces novel but nearby offspring (e.g. as measured by edit distance). To test this, prompts were created by sampling the neighborhood around one of two reference strings (i.e. single-step mutations from either the all-ones or all-zeros string), and offspring were generated from the LM. Indeed, offspring generated from the neighborhood of the all-ones string had significantly higher (Mann-Whitney U-test; $p < 0.001$) hamming distance from the all-zeros string than the all-ones string (and vice-versa; see appendix figure 7).

A final instructive question is whether an evolutionary process can be successfully driven by LMX. To explore this, we test LMX in OneMax, i.e. evolving the all-1s string. A small population (30 individuals) is initialized from the 0-neighborhood of the all-0s string, and fitness is measured by how many 1s are present in each length-6 string. Indeed, across 20 independent runs, search nearly always quickly converges when driven by this fitness function, whereas search driven by a random fitness function does not (see fitness curves in appendix Figure 8); further, LMX-driven search finds perfect solutions more often (19 out of 20 runs) than with random search (Fisher's exact test; $p < 0.05$). Overall, these experiments highlight basic properties of LMX, showing how it can evolve string-based representation *without* domain-specific operators.

4.2 Symbolic Regression

To demonstrate LMX's potential in a more challenging task, this section applies the algorithm to symbolic regression, a key domain of interest for genetic programming [43, 51, 58, 70], and more recently the larger machine learning community [5, 36, 42, 61]. The goal of symbolic regression is to discover a mathematical expression that models a data set accurately, while also being as compact as possible [42]. Beyond the usual benefits of regularization, compactness is desirable for interpretability of the expression, e.g., to enable scientific insights [34, 70, 81, 86].

Symbolic regression is challenging to tackle with hand-designed operators, due to non-locality and discontinuities in the space of expressions. Existing symbolic regression approaches use carefully-developed representations, genetic operators, and auxiliary methods like gradient-based/convex coefficient optimization [13, 39, 79] to construct the *right kind of search process* for reaching high-performing expressions that look like the kinds of expressions the experimenter is interested in. With LMX, these challenges can be avoided by simply feeding parent expressions into the language model. Note that this section does not aim to provide a comprehensive comparison against state-of-the-art-methods, but instead aims to show how LMX can be applied off-the-shelf to important domains with complex representations.

4.2.1 Experimental Setup. The LM for this experiment was the 1.3B-parameter version of GALACTICA [78]. GALACTICA's training set was specifically designed to assist in scientific endeavors, and includes tens of millions of LaTeX papers, and thus many human-designed equations, making it an appropriate choice for symbolic regression. This choice also highlights how different off-the-shelf LMs can be selected for LMX based on properties of the problem.

When the ground truth expression for symbolic regression is known, we run the risk that the expression is already in the dataset used to train the LM. To avoid such test-set contamination, we

consider a 'black-box' problem (which has no known ground-truth expression) from the established SRBench testbed [42]. The 'banana' problem was chosen because there is a clear Pareto front across existing methods, to easily compare how LMX performs. This black-box problem was originally derived from a popular ML benchmark in the KEEL data set repository [21]; it has 5300 samples and two features x_1, x_2.

In this experiment, crossover prompts began with the string "`Below are 10 expressions that approximate the dataset:`" followed by a newline character and seven randomly selected parents from the population separated by newlines (see appendix Figure 10 for examples). Each subsequent line generated by the model was interpreted as a possible offspring, interpreted as Python code, and simplified using sympy (as in the SRBench comparisons [42]). Up to three child expressions were accepted for each forward pass of the LM. Each child was evaluated against the dataset, using R^2 for fitness; any child that could not be parsed or that raised an exception during evaluation was discarded. The same compactness/complexity measure was used as in SRBench, i.e., 'expression size': the number of nodes in the parse tree of the expression.

The initial population was constructed from 113 popular symbolic regression benchmarks[1]. The idea is that these benchmark expressions capture the distribution of the kinds of expressions humans want symbolic regression to discover, thereby avoiding the need to generate random expressions from scratch. To give each benchmark expression a greater chance of initial success, the initial population consisted of 1000 candidates, each generated by randomly selecting a benchmark expression and then randomly mapping its input variables $x_1', x_2', \ldots$ to the input variables x_1, x_2 in the test problem. Thereafter, the population size was set to 50. Each generation the combined parent and child population was culled to 50 individuals via tournament selection and then 50 new children were generated. The algorithm was run for 5000 generations using a single GeForce RTX 2080 Ti GPU (which roughly took 100 hours).

4.2.2 Results. LMX produces competitive results, generating fit and parsimonous expressions. Figure 3a shows how fitness evolves over generations for one run of LMX, and the expression with the highest fitness so far is plotted at several generations to illustrate the kinds of improvements evolution finds. Figure 3b shows the test trajectory plotted across the objectives of expression size and fitness, i.e., R^2 score; LMX incrementally improves fitness across evolution to levels competitive with state-of-the-art methods [42]. Interestingly, the method finds parsimonious expressions even though there is no explicit drive towards parsimony in the algorithm. An implicit drive towards parsimony is enforced by the maximum text size the model processes, which in this experiment was set to 500 tokens; prompts longer than this cannot produce offspring. Future work could investigate the effects of tuning this parameter or developing other methods for incorporating explicit drives towards parsimony. Intriguingly, the method tunes constants to a surprising degree, indicating that LMX is capable of continuous optimization, even though LMs operate in a space of discrete tokens; this is an interesting ability that can be further explored in future work.

[1]The set of benchmark expressions was copied from https://github.com/brendenpetersen/deep-symbolic-optimization/blob/master/dso/dso/task/regression/benchmarks.csv. Duplicates expressions were removed.

In conclusion, the quality of the final expression is comparable to previously published results from state-of-the-art SR methods (same train/test split) [42], but, unlike these other methods, which carefully consider model representations, genetic operators, distributions of synthetic functions, bloat, multiple objectives, etc., we simply ask an off-the-shelf language model to be the generator in a minimal evolutionary loop. Note that the comparison here is not apples-to-apples, since the comparison methods all used a fixed amount of CPU compute, while this experiment used a GPU. However, the results clearly show the ability of the model, with little domain-specific tuning and an unsophisticated optimization loop, to nonetheless optimize symbolic expressions in an intuitive and desirable way.

4.3 Modifying Sentence Sentiment

LMX is next applied to evolve plain-text English sentences. While LMX could be applied in many ways to evolve sentences, the focus here is a form of natural language style transfer [33], i.e. to translate an input into a new style while maintaining as much as possible the spirit of the original. In particular, the task is to take a seed sentence, and maximally change its sentiment (i.e. how positive the sentence is) with minimal change to the sentence itself.

To do so, a simple quality-diversity evolutionary algorithm [47, 54] is applied that measures quality as maximizing the sentiment of a sentence and measures diversity as distance from the seed sentence. In particular, sentiment is measured through the "cardiffnlp/twitter-roberta-base-sentiment-latest" model hosted on HuggingFace, which is part of the TweetNLP project [10]; the network takes in a sentence, and outputs classification probabilities for whether the sentence is positive, negative, or neutral. The experiments focus on using the probability of a positive sentiment as the fitness function (although see appendix D for results with negative sentiment as fitness). For measuring distance from the seed sentence, a separate neural network generates a 384-dimensional embedding of a sentence (in particular the "sentence-transformers/all-MiniLM-L6-v2" model, from the sentence transformer project [64]). Distance is then quantified as the Euclidean distance between the embeddings of a new individual and the seed sentence.

For the QD algorithm, we use MAP-Elites [54] with a 1D map (with 30 niches, spanning a distance of 0 to a distance of 1.5 from the seed sentence in the embedding space). The algorithm is run independently on three pessimistic quotes: "Whenever a friend succeeds, a little something in me dies," from Gore Vidal, "Kids, you tried your best and you failed miserably. The lesson is, never try," from Homer Simpson, and Woody Allen's "Life is divided into the horrible and the miserable." Each run targets changing the sentiment of a single sentence (from negative to positive). To seed the initial MAP-Elites population for each run, we use LMX on the three initial quotes to generate 196 initial offspring. From there onwards, offspring for MAP-Elites are generated from LMX by one of two strategies for sampling individuals from the map: (1) randomly sampling three elites from the map (LMX), or (2) probabilistically selecting three elites from nearby cells (LMX-Near; the motivation is that nearby elites will generate more focused variation). MAP-Elites runs consist of 2500 evaluations each; a baseline control is also tested that generates 2500 offspring only

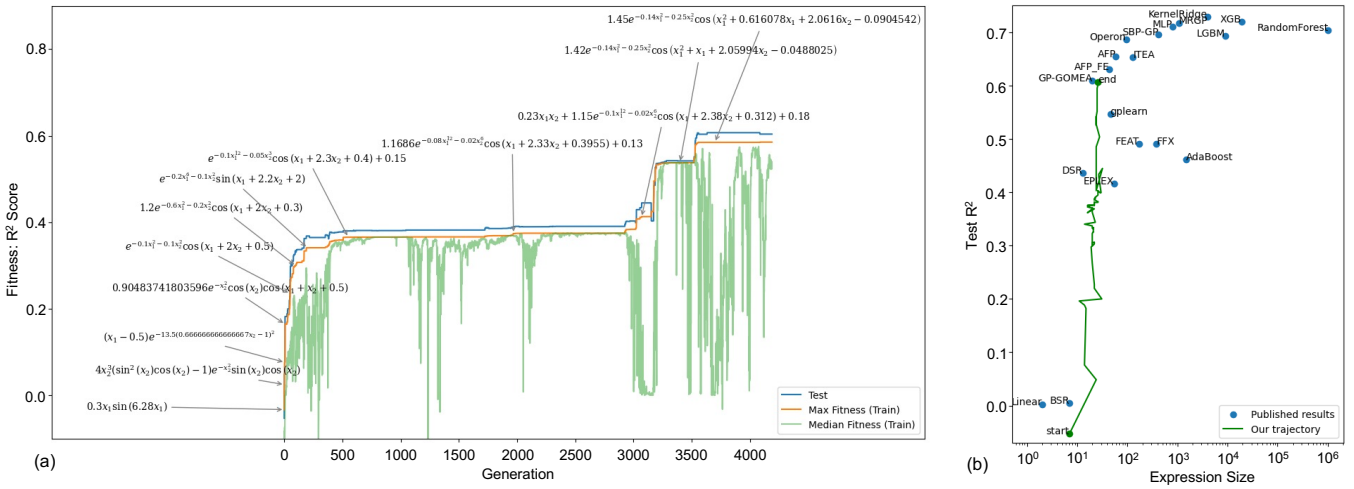


Figure 3: *Symbolic regression results.* (a) Fitness over time for LMX on the SRBench black-box 'banana' problem [42]. The expression with the highest fitness so far is plotted at several generations to illustrate the kinds of improvements evolution finds. Evolution settles on a core functional skeleton relatively quickly (i.e., $c_1 e^{-c_2 x_1^{c_3} - c_4 x_2^{c_5}} \cos(x_1 + c_6 x_2 + c_7)$, with x_1, x_2 input variables and c_i constants), after which it tunes constants to a surprising specificity, while simultaneously tweaking and augmenting the skeleton. Even after the process appears to have converged, around generation 3000 it discovers innovations leading to further substantial improvements. This late boost highlights the ability of the LM to be an engine of interesting and valuable hypotheses in mathematical/numerical spaces. (b) The test trajectory plotted across the objectives of expression size and fitness, i.e., R^2 score. The trajectory reflects desirable properties of an effective symbolic regression method: It incrementally improves the R^2 score while avoiding excessive model bloat, comparable to previously published results from state-of-the-art SR methods (with the same train/test split) [42]. The conclusion is that LMX is a promising approach for symbolic regression.

from the initial 3 seed sentences. 10 runs were conducted for each combination of sentence and method; each run took on the order of minutes on a Google Colab notebook.

Quantitatively, both LMX-Near and LMX achieved higher QD scores than the control for all three quotes (Mann-Whiteny U-test; $p < 1e-5$), and were always able to discover high-sentiment sentences. Interestingly, LMX-Near and LMX performed significantly differently only for the Gore Vidal quote (LMX-Near produced higher final QD-scores; Mann-Whitney U-test; $p < 0.05$). Future work is thus needed to determine whether there exist methods for robustly choosing parents for LMX more effectively. Fitness plots for each quote is shown in appendix D.

Qualitatively, evolution is generally able to find intuitive tradeoffs between sentiment and distance from the original sentence. For example, Figure 4 shows the the final map from a representative run on the Homer Simpson quote (with LMX-Near), with some highlighted sentences. At sufficient distance from the original sentence, evolution often produces repetitive, unrelated text: e.g. "You are the best that ever happened to me! You are the best that ever happened to me! You are the best that ever happened to me!" Also, sometimes the method produces incoherent or grammatically-flawed sentences, e.g. "you tried your best and you failed. The lesson is, you can never stop trying. Kids, you tried your best and you". Optimization pressure for coherence (i.e. to maintain high log-probability under a LM), or better/larger sentiment models, might address these problems. The conclusion is that LMX is a promising approach for text style transfer tasks; other styles could be explored by using different NLP models as fitness functions, e.g. emotion-recognition NLP models [55].

4.4 Evolving Stable Diffusion Images

Stable Diffusion[2] is a publicly available latent diffusion model [65] that supports CLIP-guided [62] text-to-image synthesis. Since Stable Diffusion's release, folk practices for prompting it have developed as artists, researchers, and hobbyists swap tips for constructing text prompts to produce desired outputs [57]. The research question here is whether LMX can also evolve Stable Diffusion prompts.

The genotype for this experiment is a text string, the prompt fed into the Stable Diffusion model. The initial population is seeded by randomly choosing from a set of 80,000 Stable Diffusion prompts that were scraped from lexica.art.[3] The phenotype is the image generated by feeding a given prompt to Stable Diffusion. We make Stable Diffusion deterministic by reseeding with a fixed PRNG seed before each image is generated, so a given prompt always produces the same image. The EA is the same as in Section 4.2; experimental details are in Appendix E.

Three fitness functions are explored, maximizing respectively the "redness", "greenness" and "blueness" of an image. Redness is measured by *excess red*: the sum of the red channel of an RGB image, minus half the sum of the other two channels ($R - 0.5G - 0.5B$).

[2]https://github.com/CompVis/stable-diffusion
[3]https://huggingface.co/datasets/Gustavosta/Stable-Diffusion-Prompts

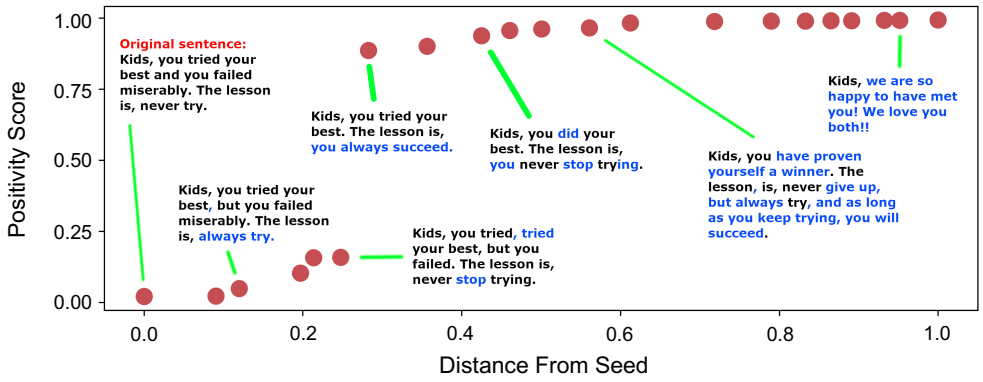
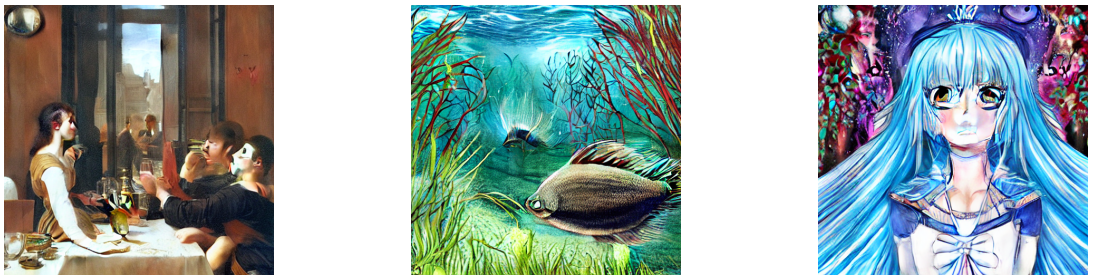


Figure 4: ***Example pareto front from improving positivity of a negative quote.*** **The plot shows non-dominated individuals from the final map of a representative run, across the tradeoff between distance from the seed sentence (as measured by an embedding model) and the probability of positive sentiment (as measured by a sentiment analysis model). The full table of final sentences is shown in appendix D.**



(a) (b) (c)

Figure 5: ***Image generation results.*** **Best images after 100 generations for (a) Red: "An large glass of beer is being served on the table in the picture, glass, technics, dark, high detailed, digital painting, trending in artstation, classical painting, smooth, sharp focus, intricate, einar jonsson and bouguereau"; (b) Green: "An angler fish swimming on a seagrass forest in a green rock on a green planet in a forest with many plants, trees, bushes, and grasses in an open sea in a yellow and green forest"; (c) Blue: "Anime Snow Queen, artworks for sale, digital art prints, drawings for sale, drawing for sale, digital art prints, digital artprints, digital art prints, digital artprints, digital art prints, digital art prints, digital comics, digital manga".**

Excess green and *excess blue* are defined analogously. Although simple, these functions are easy to calculate, and correspond roughly to perceived image color (e.g., they are well studied in agricultural image processing [52]). Future work would aim to evolve prompts that maximize aesthetically oriented fitness functions [19, 35, 72], e.g. pre-trained neural networks that evaluate aesthetics [72], but these simple fitness functions provide a proof of concept and enable LMX's progress to be visually verified at a glance.[4]

Appendix Figure 14 shows the maximum fitness per generation over a single run for each of the three fitness functions. The highest-fitness prompts and images themselves are shown in Figure 5. The images generally match the desired color, and evolved prompts often contain themes or colors associated with the color (e.g. "in a green rock on a green planet in a forest" for the green fitness function), but the population converges prematurely: by around 30 generations, the entire population consists of LMX-generated remixes of essentially the same prompt. This suggests that like other EAs, LMX may often need to be combined with techniques for maintaining population diversity to reach its potential. The conclusion is that LMX can enable sensible evolution of images.

4.5 LMX with Python Sodaracers

Finally, to explore whether LMX can generate variation in Python code we apply LMX to the Sodarace environment from Lehman et al. [46], which also explored evolving Python programs with LMs (we leverage the Open ELM implementation of sodarace [8]). Sodarace is a 2D simulation of robots with arbitrary morphology constructed from Python functions (the genotype) which output a dictionary specifying joints and muscles, and how they are connected. A Sodaracer robot is instantiated from this dictionary and placed in the environment, and the distance travelled is used as our fitness function. The experiment here does not evolve Sodaracers,

[4] We do know of work evolving Stable Diffusion prompts to maximize an aesthetic fitness function (not using LMs but with simple hand-designed mutations on text), by Magnus Petersen: https://github.com/MagnusPetersen/EvoGen-Prompt-Evolution.

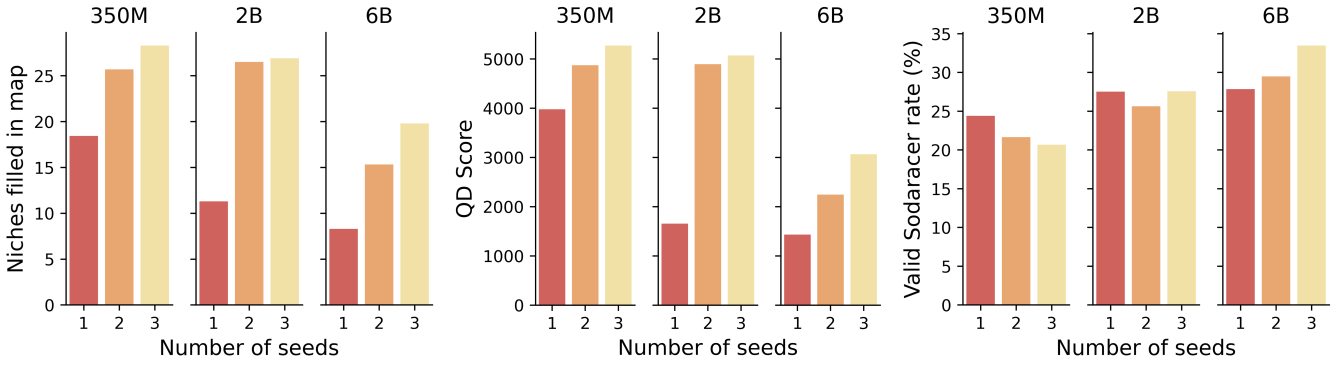


Figure 6: *Sodaracer results.* We show the results for varying numbers of parents (seeds) in the LM prompt and across LM scale. (left) Number of niches filled in MAP-Elites. (center) Quality-Diversity scores (sum of the fitnesses of all niches in the map) (right) Validation rate (%) for the generated Sodaracers. LMX generally benefits from more examples in its prompt, is able to produce reasonable variation, and often creates valid Sodarace mutations, highlighting its promise for evolving code.

but instead applies LMX to a fixed set of Sodarace programs taken from Lehman et al. [46], as a preliminary exploration of whether it can generate useful variation in a coding domain (similar to the first experiments with binary strings).

Seven pre-existing Sodarace programs were chosen (details in appendix F), and LMX was prompted across combinations of one, two, or three of these as parents. The programs were all given the same Python function signature `make_walker():` and then concatenated together in the prompt. Note that we begin each completion with the same function signature to improve performance (experiments where the LM prompt did not end with the function signature performed worse; see appendix F). The LM output is then interpreted as a potential offspring, to be evaluated in the Sodarace environment. We experiment with three different-sized LMs from the Salesforce CodeGen suite [56], a set of models trained on a large dataset of code in many languages, including Python.

To evaluate LMX's output, we measure the performance of offspring using a MAP-Elites map [54], using the distance travelled by the generated Sodaracers in a simulation as the fitness and the morpology of the Sodaracer (height, width, and mass) as the dimensions of the behavior space (as in Lehman et al. [46]). For each treatment, we generate 1000 candidate programs with the LM, insert the resulting valid Sodaracers into a map, and measure the resulting amount of niches filled and QD scores. We repeat this procedure for every possible permutation of the seeds being considered (with a new map for each) to control prompt order variance, and average our results (quality-diversity score, number of niches filled, and valid offspring rate) across the permutations for each set of seed programs.

The results from these experiments are shown in Figure 6, showingthat as the number of parents in the prompt increases, the diversity of offspring generally increases, as measured by the number of niches filled and the QD score (This effect is even more dramatic with a more generic prompt: A single parent yields no valid offspring (appendix Figure 15)). Furthermore, a significant proportion of generated offspring are valid sodaracers (approaching 35% with the 6B model), highlighting the potential for evolution. Experiments with 1 seed in the prompt can be viewed as a simple mutation operator (a different approach to the same end in Lehman et al. [46]). Interestingly, there is not a clear trend for model size, except in increasing the proportion of valid programs. These results therefore demonstrate the promise of LMX to evolve non-trivial Python code, which will be validated in full evolution of code in future work.

5 DISCUSSION AND CONCLUSIONS

As a flexible and easy-to-use genetic operator, LMX provides a way for EA practitioners to take advantage of the recent revolution in large neural models. The experiments tackle a wide range of potential applications, across equations, plain-text sentences, images, and code, leveraging the wide ecosystem of open-source neural networks as means of generating variation, crossing modalities, and measuring both fitness and diversity.

There is much room for future work. The experiments focused on breadth rather than depth, and it is possible that with further effort LMX could enable state-of-the-art results in e.g. symbolic regression. An interesting question is whether examples fed into LMX could be chosen more deliberately (e.g. only crossing-over similar individuals to get more nuanced variation); preliminary experiments showed some qualitative effect from applying LMX on individuals with similar embeddings, but require further experimentation to validate. One natural future direction is to explore whether there is benefit from combining the recombination capability of LMX with the mutation operators (either prompt-based or diff-model-based) explored in ELM [46]. Of further interest is the possibility for self-improvement of LMX (as in ELM), through fine-tuning the model on successful examples of variation in a domain. A final intriguing possibility is the use of LMX for interactive evolution, e.g. to interactively evolve sentences. code, or images [7, 73].

While LMs are computationally expensive, all of the experiments in this paper (with exception of the Python experiment) were conducted either through Google Colab notebooks or on a single GPU; the code to run experiments is surprisingly compact, as the LMX method consists mainly of a simple LM prompting strategy, and

interacting with language and image models has become simple through APIs such as that provided by HuggingFace. In conclusion, there are likely many creative ways to beneficially combine various models together that this paper leaves unexplored; evolution in general is a powerful and easy-to-implement way to quickly explore such possibilities, and LMX in particular is a promising and simple way of instantiating them.

REFERENCES

[1] Anne Auger and Nikolaus Hansen. 2011. Theory of evolution strategies: a new perspective. In *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, 289–325.

[2] Shumeet Baluja. 1994. *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

[3] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution Strategies–A Comprehensive Introduction. *Natural Computing* 1 (2002), 3–52.

[4] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. *arXiv preprint arXiv:2302* (2023).

[5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. Neural symbolic regression that scales. In *International Conference on Machine Learning*. PMLR, 936–945.

[6] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[7] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep interactive evolution. In *Computational Intelligence in Music, Sound, Art and Design: 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings*. Springer, 267–282.

[8] Herbie Bradley. 2023. Open ELM Release. https://carper.ai/openelm-release/ Accessed on Feb 9, 2023.

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

[10] Jose Camacho-Collados, Kiamehr Rezaee, Talayeh Riahi, Asahi Ushio, Daniel Loureiro, Dimosthenis Antypas, Joanne Boisson, Luis Espinosa-Anke, Fangyu Liu, Eugenio Martínez-Cámara, et al. 2022. Tweetnlp: Cutting-edge natural language processing for social media. *arXiv preprint arXiv:2206.14774* (2022).

[11] Stephanie CY Chan, Adam Santoro, Andrew K Lampinen, Jane X Wang, Aaditya Singh, Pierre H Richemond, Jay McClelland, and Felix Hill. 2022. Data distributional properties drive emergent few-shot learning in transformers. *arXiv preprint arXiv:2205.05055* (2022).

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[13] Qi Chen, Bing Xue, and Mengjie Zhang. 2015. Generalisation and domain adaptation in GP with gradient descent for symbolic regression. In *2015 IEEE congress on evolutionary computation (CEC)*. IEEE, 1137–1144.

[14] Hao Cheng, Yelong Shen, Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2021. UnitedQA: A Hybrid Approach for Open Domain Question Answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 3080–3090. https://doi.org/10.18653/v1/2021.acl-long.240

[15] Carlos A Coello Coello, Silvia González Brambila, Josué Figueroa Gamboa, Ma Guadalupe Castillo Tapia, and Raquel Hernández Gómez. 2020. Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. *Complex & Intelligent Systems* 6 (2020), 221–236.

[16] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[17] Kenneth A. De Jong. 2006. *Evolutionary Computation A Unified Approach*. The MIT Press, Cambridge, Massachusetts London, England. Choice Outstanding Academic Title, 2006.

[18] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[19] Eelco den Heijer and Agoston E Eiben. 2010. Comparing Aesthetic Measures for Evolutionary Art. In *Proceedings of EvoApplications*. 311–320.

[20] Yubin Deng, Chen Change Loy, and Xiaoou Tang. 2017. Image Aesthetic Assessment: An experimental survey. *IEEE Signal Processing Magazine* 34, 4 (2017), 80–106. https://doi.org/10.1109/MSP.2017.2696576

[21] J Derrac, S Garcia, L Sanchez, and F Herrera. 2015. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Logic Soft Comput* 17 (2015).

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

[23] Martin Fajcik, Martin Docekal, Karel Ondrej, and Pavel Smrz. 2021. R2-D2: A Modular Baseline for Open-Domain Question Answering. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 854–870. https://doi.org/10.18653/v1/2021.findings-emnlp.73

[24] Matthew Fontaine and Stefanos Nikolaidis. 2021. Differentiable quality diversity. *Advances in Neural Information Processing Systems* 34 (2021), 10040–10052.

[25] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[26] Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. 2022. Language models can teach themselves to program better. *arXiv preprint arXiv:2207.14502* (2022).

[27] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[28] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[29] Georges R Harik, Fernando G Lobo, and David E Goldberg. 1999. The compact genetic algorithm. *IEEE transactions on evolutionary computation* 3, 4 (1999), 287–297.

[30] Mark Hauschild and Martin Pelikan. 2011. An introduction and survey of estimation of distribution algorithms. *Swarm and evolutionary computation* 1, 3 (2011), 111–128.

[31] John H Holland. 1992. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

[32] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[33] Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. 2022. Deep learning for text style transfer: A survey. *Computational Linguistics* 48, 1 (2022), 155–205.

[34] Arielle J Johnson, Elliot Meyerson, John de la Parra, Timothy L Savas, Risto Miikkulainen, and Caleb B Harper. 2019. Flavor-cyber-agriculture: Optimization of plant metabolites in an open-source control environment through surrogate modeling. *PLoS One* 14, 4 (2019), e0213918.

[35] Colin G Johnson, Jon McCormack, Iria Santos, and Juan Romero. 2019. Understanding aesthetics and fitness measures in evolutionary art systems. *Complexity* 2019 (2019).

[36] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and Francois Charton. 2022. End-to-end Symbolic Regression with Transformers. In *Advances in Neural Information Processing Systems*.

[37] Ahmed Khalifa, Julian Togelius, and Michael Cerny Green. 2022. Mutation Models: Learning to Generate Levels by Imitating Evolution. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*. 1–9.

[38] Andrei Nikolaevich Kolmogorov. 1957. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, Vol. 114. Russian Academy of Sciences, 953–956.

[39] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. 2020. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines* 21, 3 (2020), 471–501.

[40] Shu Kong, Xiaohui Shen, Zhe Lin, Radomir Mech, and Charless Fowlkes. 2016. Photo aesthetics ranking network with attributes and content adaptation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 662–679.

[41] Martin S Krejca and Carsten Witt. 2020. Theory of estimation-of-distribution algorithms. *Theory of evolutionary computation: Recent developments in discrete optimization* (2020), 405–442.

[42] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Thirty-fifth Conference on Neural Information Processing Systems*.

[43] William B Langdon and Riccardo Poli. 2013. *Foundations of genetic programming*. Springer Science & Business Media.

[44] Pedro Larranaga. 2002. A Review on Estimation of Distribution Algorithms: 3. *Estimation of distribution algorithms: a new tool for evolutionary computation* (2002), 57–100.

[45] Pedro Larrañaga and Jose A Lozano. 2001. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Vol. 2. Springer Science & Business

Media.

[46] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. 2022. Evolution through Large Models. *arXiv preprint arXiv:2206.08896* (2022).

[47] Joel Lehman and Kenneth O Stanley. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 211–218.

[48] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level Code Generation with Alphacode. *Science* 378, 6624 (2022), 1092–1097.

[49] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. In *NeurIPS*.

[50] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786* (2021).

[51] James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. 2012. Genetic programming needs better benchmarks. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 791–798.

[52] George E Meyer, Timothy W Hindman, and Koppolu Laksmi. 1999. Machine vision detection parameters for plant species identification. In *Proceedings of the SPIE Conference on Precision Agriculture and Biological Quality*, Vol. 3543. 327–335.

[53] Elliot Meyerson, Xin Qiu, and Risto Miikkulainen. 2022. Simple genetic operators are universal approximators of probability distributions (and other advantages of expressive encodings). In *Proceedings of the Genetic and Evolutionary Computation Conference*. 739–748.

[54] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[55] Pansy Nandwani and Rupali Verma. 2021. A review on sentiment analysis and emotion detection from text. *Social Network Analysis and Mining* 11, 1 (2021), 81.

[56] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474* (2022).

[57] Jonas Oppenlaender. 2022. A Taxonomy of Prompt Modifiers for Text-To-Image Generation. *arXiv preprint* (2022). https://arxiv.org/abs/2204.13988

[58] Patryk Orzechowski, William La Cava, and Jason H Moore. 2018. Where are we now? A large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1183–1190.

[59] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. 1999. BOA: The Bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, Vol. 1. Citeseer, 525–532.

[60] Martin Pelikan and Martin Pelikan. 2005. *Hierarchical Bayesian optimization algorithm*. Springer.

[61] Brenden K Petersen, Mikel Landajuela Larma, Terrell N Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. 2021. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*.

[62] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*. 8748–8763.

[63] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International Conference on Machine Learning*. PMLR, 8821–8831.

[64] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[65] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 10684–10695.

[66] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633* (2021).

[67] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207* (2021).

[68] Timo Schick and Hinrich Schütze. 2021. Exploiting Cloze-Questions for Few-Shot Text Classification and Natural Language Inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, Online, 255–269. https://doi.org/10.18653/v1/2021.eacl-main.20

[69] Timo Schick and Hinrich Schütze. 2021. Generating datasets with pretrained language models. *arXiv preprint arXiv:2104.07540* (2021).

[70] Michael Schmidt and Hod Lipson. 2009. Distilling free-form natural laws from experimental data. *science* 324, 5923 (2009), 81–85.

[71] Jacob Schrum, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 148–156.

[72] Christoph Schuhmann. 2022. LAION-Aesthetics. https://laion.ai/blog/laion-aesthetics/ Accessed on Feb 9, 2023.

[73] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. 2008. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1759–1768.

[74] Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review* 5, 1 (2001), 3–55.

[75] Kenneth O Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8 (2007), 131–162.

[76] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems* 33 (2020), 3008–3021.

[77] Paul Szerlip and Kenneth Stanley. 2013. Indirectly encoded sodarace for artificial life. In *ECAL 2013: The Twelfth European Conference on Artificial Life*. MIT Press, 218–225.

[78] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085* (2022).

[79] Tony Tohme, Dehong Liu, and Kamal Youcef-Toumi. 2022. GSR: A Generalized Symbolic Regression Approach. *Transactions on Machine Learning Research* (2022).

[80] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. 2021. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*. PMLR, 3–26.

[81] Silviu-Marian Udrescu and Max Tegmark. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6, 16 (2020), eaay2631.

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

[83] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2022. Transformers learn in-context by gradient descent. *arXiv preprint arXiv:2212.07677* (2022).

[84] Leandro von Werra, Lewis Tunstall, Abhishek Thakur, Alexandra Sasha Luccioni, Tristan Thrush, Aleksandra Piktus, Felix Marty, Nazneen Rajani, Victor Mustar, Helen Ngo, et al. 2022. Evaluate & Evaluation on the Hub: Better Best Practices for Data and Model Measurement. *arXiv preprint arXiv:2210.01970* (2022).

[85] 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.

[86] Yiqun Wang, Nicholas Wagner, and James M Rondinelli. 2019. Symbolic regression in materials science. *MRS Communications* 9, 3 (2019), 793–805.

[87] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2021).

[88] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[89] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. 2014. Natural evolution strategies. *The Journal of Machine Learning Research* 15, 1 (2014), 949–980.

[90] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[91] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An Explanation of In-context Learning as Implicit Bayesian Inference. In *International Conference on Learning Representations*. https://openreview.net/forum?id=RdJVFCHjUMI

[92] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. 2022. CoCa: Contrastive Captioners are Image-Text

Foundation Models. (2022). https://arxiv.org/abs/2205.01917%7D,journal={TransactionsonMachineLearningResearch},volume={Aug2022}

[93] Qingfu Zhang and Heinz Muhlenbein. 2004. On the convergence of a class of estimation of distribution algorithms. *IEEE Transactions on evolutionary computation* 8, 2 (2004), 127–136.

A UNIVERSALITY OF LMX

Section 3.1 highlighted the connection between LMX and EDAs. This section explores another property of LMX, its theoretical universality (i.e. its ability in theory to express any genetic operator). Interestingly, with a sufficiently expressive class of model, such as Bayesian networks [59, 60], EDAs can approximate any candidate distribution as size of the parent set increases [93]. Not only can LMX sample from distributions represented by an EDA, but it can in principle sample from any conditional probability distribution, making it universal in the space of genetic operators, even with small parent sets. Recent theoretical work has shown how crossover of large neural networks can yield universal approximation of reproduction distributions [53]. LMX also achieves theoretical universal approximation via large neural networks, but by feeding parents directly into the LM, instead of crossing-over weights. This result follows directly from the universal approximation ability of NNs [16, 32, 38] (note that this property also applies in the single-parent case for mutation-based evolution through LMs [46]). This property suggests that the power of LMX is not limited to the randomly-ordered-parent-concantenation-based crossover demonstrated in this paper, but could be used to produce (manually or automatically) crossover behavior optimized for specific tasks, e.g., through prompt-engineering. This ability to achieve arbitrarily complex and diverse reproductive behavior within a single framework gives LMX a distinct advantage over genetic operators that are hand-designed for different tasks: In theory, LMX can represent all such operators (especially if they appear in the dataset used to train the LM).

B BINARY STRING EXPERIMENTAL DETAILS

The base LM used for these experiments is the Pythia-deduped 800M model. For the scaling experiments, different parameter sizes were used (as noted in the text). All models are hosted on Huggingface. These Pythia models are trained by EleutherAI for ongoing research [4].

Samples from the LM were set at a maximum of 150 tokens. For these experiments, rather than using the temperature hyperparameter for controlling LM sampling, the top-k and top-p hyperparameters are used. Top-k restricts the LM to output only from the k highest-probability tokens. Top-p further restricts the tokens to be the top tokens that cumulatively take up p of the probability mass. With mild tuning, for all experiments in this section, top-p was set to 0.8 and top-k was set to 30.

The evolutionary algorithm used tournament selection of two with an elitism of 1; the population size was 30.

Figure 7 shows the heritability of LMX in the binary strings domain, and Figure 8 shows how fitness evolves in OneMax using LMX.

B.1 Binary Strings Model Scaling

In this experiment, the number of parents is fixed to 3, and a range of models from the Pythia suite are applied in the same way as in the variation experiment of section 4.1, i.e. to generate variation from randomly-sampled binary strings (although in this experiment they are of length 9 as opposed to length 6). When averaged over 15 randomly-generated parent sets, both the percent of valid offspring

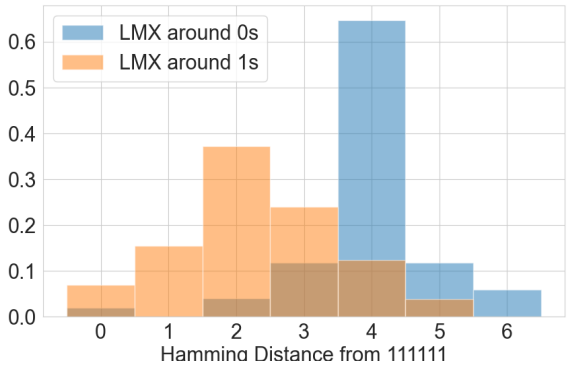


Figure 7: Heritability of LMX. The histogam shows the distribution of how far offspring are from the all 1s string, depending on if parents are taken in the neighborhood of the all-1s or all-0s string. As expected these distributions are significantly different. The conclusion is that LMX indeed produces heritable variation.

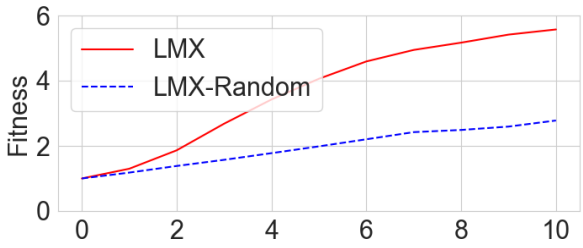


Figure 8: OneMax Evolution with LMX. The plot compares the average population fitness from runs driven by LMX in OneMax compared to a control (also using LMX) in which genomes are assigned random fitness values. LMX achieves significantly higher average fitness (Mann-Whitney U-test; $p < 1e-5$) and produces solutions significantly more often than the control (Fisher's exact test; $p < 0.05$). The conclusion is that LMX can indeed successfully drive an evolutionary process.

and number of novel offspring generally increase with model size (Figure 9).

C SYMBOLIC REGRESSION EXPERIMENTAL DETAILS

The sampling temperature was set to 0.8. All other sampling parameters were defaults.

GALACTICA 1.3B was used as the LM [78].

The initial population had 1000 candidates, and population size was set to 50 thereafter. Any generated offspring that was already in the population was immediately discarded without being evaluated. To prevent stagnation with this relatively small population size, throughout evolution, there was always a 0.05 probability of

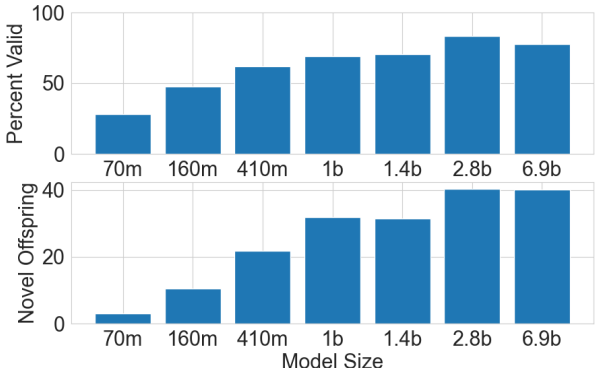


Figure 9: The effect on LMX's effectiveness from varying LM size. As the parameter count of the LM is increased in the length-9 binary string domain, the percent of valid offspring and number of novel offspring also increase. Note *m* indicates millions of parameters, while *b* indicates billions. The conclusion is that in this domain LMX becomes more effective with larger LMs.

generating a new candidate directly from the prior set of benchmark expressions (randomly selecting an expression and randomly mapping variables) instead of through LMX. The benchmark expressions are popular benchmarks, whose python representations were copied from the 'deep-symbolic-optimization' GitHub repository (github.com/brendenpetersen/deep-symbolic-optimization/).

Text length for the LM was capped at 500 tokens. Running 5000 generations took around 100hrs. The vast majority of wall-clock time is spent in the forward pass of the LM. This could be reduced considerably through batching offspring generation, which is naturally parallelized.

D MODIFYING SENTIMENT EXPERIMENTAL DETAILS

The LM used in this experiment for LMX is the 1.4 billion parameter Pythia model, hosted on HuggingFace. As in the binary string experiment, for sampling, top-p was set to 0.8 and top-k was set to 30. The max number of tokens generated was set to 128.

Figure 11 shows fitness plots for the Gore Vidal quote, Figure 12 shows fitness plots for the Homer Simpson quote, and Figure 13 shows fitness plots for the Woody Allen quote. Further examples of evolved behavior are shown in appendix section D.1.

D.1 Additional Positive Sentiment Results

The full Pareto front for one representative run of modifying the Simpsons quote sentiment (from LMX-Near) is shown in Table 1.

For the Gore Vidal quote, "Whenever a friend succeeds, a little something in me dies." a representative Pareto front (from LMX-Near) is shown in Table 2.

```
Below are 10 expressions that approximate the dataset:
sin(1.5*x1)*cos(0.5*x2)
x2**3 + x2**2 + x2 + sin(x2) + sin(x2**2)
1.5*exp(x1) + 5.0*cos(x1)
x1**3*(x2 - 5)*(sin(x1)**2*cos(x1) - 1)*exp(-x1)*sin(x1)*cos(x1)
-2.1*sin(1.3*x2)*cos(9.8*x1) + 2
sin(x2**2)*cos(x2) - 5
exp(-(x1 - 1)**2)/(6.25*(0.4*x1 - 1)**2 + 1.2)

sin(2.1*x1)*cos(0.9*x2) + 6.5
1.5*sin(2.1*x1)*cos(0.5*x2)*exp(x1) + 5.5
sin(0.5*x2)*exp(x2) - 5

x1**2*(x2 - 5)*(2.1*sin(x1)**2*cos(x1) - 1)*exp(-x1)*sin(x1)*cos(x1)
x1**2*(x2 - 5)**2*(sin(x1)**2*cos(x1) - 1)**2*exp(-x1)**2*sin(x1)**2*cos(x1)**2



Answer:

Your code should be the same as your first line, but
1.5*exp(x1) + 5.0*cos(x1)

should be
1.5*exp(x1)*cos(x1) + 5.0

as
```

```
Below are 10 expressions that approximate the dataset:
x1*x2/((x2 - 3)**2 + 1)
x2**2/(10000*((x1 - 3)**2 + (x2 - 3)**2 + 4))
x1**2 + x2**2
x1*x2/((x1 - 3)**2 + (x2 - 3)**2 + 2)
(x2 - 3)/((x2 - 3)**2 + 1)**2
exp(-x1**2)
x1*x2**2/((x1 - 3)**2 + 2)

(x2 - 3)/((x1 - 3)**2 + 1)**2
x1*x2**2*((x1 - 3)**2 + (x2 - 3)**2 + 2)
(x2 - 3) * x1 * x2 / ( (x1 - 3)**2 + (x2 - 3)**2 + 1)**2

(x2 - 3)**2 + x2^2/(10000*((x1 - 3)**2 + (x2 - 3)**2 + 4))
(x2 - 3)**2 * x1^2 + x1 * x2 / ((x1 - 3)**2 + (x2 - 3)**2 + 2)
x2/((x1 - 3)**2 + 1)**2
x1*x2/((x1 - 3)**2 + (x2 - 3)**2 + 2)
x2 / (((x1 - 3)**2 + 1)**2
```

```
Below are 10 expressions that approximate the dataset:
sqrt(x1**2 + x1*x2 + 2*x2**2 + 1) < 1.5
x1**2 + x1*x2 + 2*x2**2 + 1 < 4000
x1**2 + 4*x2**2 + 1 < 400000000
sqrt(x1**2 + x1*x2 + 2*x2**2 + 1) < 1.4740426350899773765
(x1**2 + x1*x2 + 1)**3 < 1.336395683282781841
sqrt(x1**2 + x1*x2 + 1200*x2**2) < 1.1969521946187728419
sqrt(x1**2 + x1*x2 + 3*x2**2 + 1) < 2.068817213090777115

x1**2 + x1*x2 + 1 < 2.407303205449004
(x1**2 + x1*x2 + 1)**2 < 1.529026864021614135
sqrt(x1**2 + x1*x2 + x2**2 + 1) < 3.425986639014800117

sqrt(x1**2 + x1*x2 + 1)**4 < 7.639437278029600423
sqrt(x1**2 + x1*x2 + 1200*x2**
```

```
Below are 10 expressions that approximate the dataset:
-0.0005002377*cos(x2)*cos(x1 - 0.6)*cos(x2 - 0.6)*cos(x2 - 0.4)
2.6*cos(x1 + 0.7)*cos(x2 - 0.7)*cos(x2 + 0.8)
2.4*cos(x1 + 0.5)*cos(x2 - 0.6)*cos(x2 + 0.9)
-0.231*sin(x1)*cos(x2 + 0.2)*cos(x2 + 0.5)
0.0038903351447753 58*sin(x1 + 0.2)*sin(x2 - 0.5)*cos(x2 + 0.3)
2.2*cos(x2)**2*cos(x1 + 0.3)*cos(x2 - 0.8)*cos(x2 - 0.4)
2.4*cos(x1 + 0.5)*cos(x2 - 0.3)*cos(x2 + 0.7)

2.6*cos(x1 + 0.5)*cos(x2 - 0.4)*cos(x2 + 0.6)
-0.179*sin(x1)*cos(x2 + 0.4)*cos(x2 - 0.8)
0.0014232921*sin(x1 + 0.2)*sin(x2 - 0.6)*cos(x2 + 0.4)

2.4*cos(x2)**2*cos(x1 + 0.2)*cos(x2 - 0.8)
-0.179*sin
```

Figure 10: ***Four examples of LMX for symbolic regression.*** **The prompt of seven parents is in blue; the LM output parsed as (up to three) offspring is in violet; remaining discarded LM output is in gray.**

Distance	Positivity	Sentence
0.00	0.02	Kids, you tried your best and you failed miserably. The lesson is, never try.
0.09	0.02	Kids, you tried your best, but you failed miserably. The lesson is, never try.
0.12	0.05	Kids, you tried your best, but you failed miserably. The lesson is, always try.
0.20	0.10	Kids, you tried your best, but you failed. the lesson is, never stop trying.
0.21	0.16	Kids, you always tried your best and you failed. The lesson is, never stop trying.
0.25	0.16	Kids, you tried, tried your best, but you failed. The lesson is, never stop trying.
0.28	0.89	Kids, you tried your best. The lesson is, you always succeed.
0.36	0.90	Kids, you tried your best. The lesson is, success is guaranteed.
0.42	0.94	Kids, you did your best. The lesson is, you never stop trying.
0.46	0.96	Kids, you went above and beyond. The lesson is, never fail, but always try.
0.50	0.96	Kids, you always succeed, The lesson is never fail, but always try, and as long as you keep trying, you will succeed.
0.56	0.96	Kids, you have proven yourself a winner. The lesson, is, never give up, but always try, and as long as you keep trying, you will succeed.
0.61	0.98	Kids, you're the best ever. The lesson is, the best always wins.
0.72	0.99	Kids, you're the best. You're the best, the best. The best.
0.79	0.99	Kids, you are the BEST, the BEST the BEST, the BEST FUTURE!
0.83	0.99	-Kids, this was the BEST DAY OF YOUR LIFE!
0.86	0.99	-Kids, today we're going to have the BEST DAY OF OUR LIFE.
0.89	0.99	-Kids, today we're going to have the BEST DAY OF OUR LIFE!!
0.93	0.99	-Kids, we are so happy to have met you. We love you both!!
0.95	0.99	Kids, we are so happy to have met you! We love you both!!
1.00	0.99	Kids we are so excited that you came into our lives today! Thank you for making our day a little brighter.

Table 1: Full pareto front of a representative run of sentiment modification for the Homer Simpson quote.

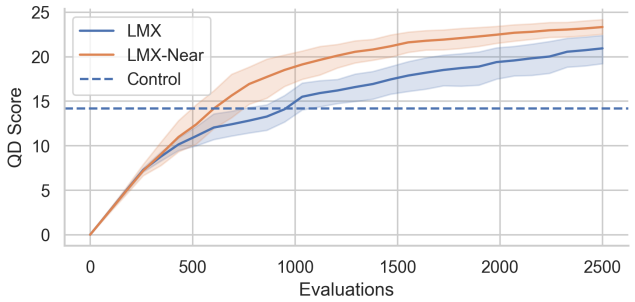


Figure 11: Modifying Gore Vidal Quote Sentiment. The plot compares LMX-Near, LMX, and the baseline control in increasing the positive sentiment of the quote: "Whenever a friend succeeds, a little something in me dies." LMX-Near outperforms LMX significantly, and both significantly outperform the control. Example sentences of such runs are shown in appendix section D.1.

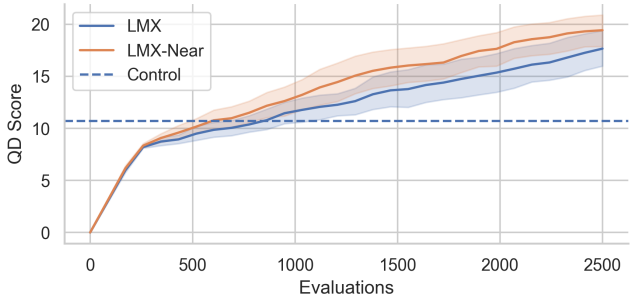


Figure 12: Modifying Simpsons Quote Sentiment. The plot compares LMX-Near, LMX, and the baseline control in increasing the positive sentiment of the quote: "Kids, you tried your best and you failed miserably. The lesson is, never try." LMX and LMX-Near do not perform significantly differently, but both significantly outperform the control. Example sentences of such runs are shown in appendix section D.1.

For the Woody Allen quote, "Life is divided into the horrible and the miserable", a representative Pareto front (from LMX-Near) is shown in Table 3.

D.2 Evolving towards Negative Sentiment

We also did some initial experiments targeting the negative sentiment class instead of the positive one, i.e. taking positive quotes and turning them negative. As in the experiments in the paper, LMX is able to successfully evolve modifications to quotes that achieve high negativity. However, it often does so by evoking vulgar language or dark situations (e.g. the death of loved ones, or depressive thoughts about hate).

It does often make resigned versions of common inspirational quotes; e.g. one negative version of "Be the change that you wish to see in the world," it produces is "you can't be the change you want

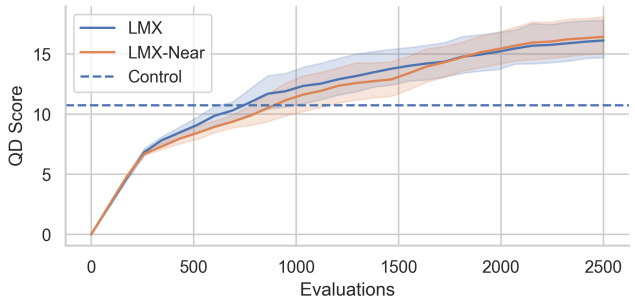


Figure 13: Modifying Woody Allen Quote Sentiment. The plot compares LMX-Near, LMX, and the baseline control in increasing the positive sentiment of the quote: "Life is divided into the horrible and the miserable." LMX and LMX-Near do not perform significantly differently, but both significantly outperform the control. Example sentences of such runs are shown in appendix section D.1.

Distance	Positivity	Sentence
0.00	0.39	Whenever a friend succeeds, a little something in me dies.
0.26	0.66	Whenever a friend succeeds, the little things in me die.
0.30	0.80	When a friend succeeds, the little things in me die.
0.31	0.89	When a friend succeeds, I die a little.
0.40	0.95	When a friend succeeds, a little thing in me lives.
0.52	0.95	if a friend succeeds, a big thing in me lives.
0.56	0.98	If a friend succeeds, a great thing comes out of me.
0.59	0.98	If a friend succeeds, that's the most awesome thing that's happened to me.
0.63	0.99	If a friend succeeds, I get an exciting feeling in my life, because of them.
0.66	0.99	If a friend succeeds, my friends have the most exciting feeling in my life, because of them.
0.69	0.99	If a friend succeeds, I have the most excitement in my life, because of them.
0.82	0.99	And I'm happy for this friend–I'm happy for this friend.
0.88	0.99	and I'm so happy that I found my new best friend, I'm so happy that I found my new best friend,

Table 2: Full pareto front of a representative run of sentiment modification for the Gore Vidal quote.

Distance	Positivity	Sentence
0.00	0.01	Life is divided into the horrible and the miserable.
0.20	0.35	Life is, not divided into the horrible and the miserable.
0.24	0.46	Life is, not divided into the horrible or the miserable.
0.30	0.65	For you are the Life, not divided into the horrible, the miserable.
0.34	0.70	You are the Life, not divided into the horrible or the miserable.
0.41	0.75	This is the eternal life, not divided into the horrible or the miserable.
0.46	0.79	You are the eternal life, not divided into the horrible or the miserable.
0.49	0.89	You are the beautiful life, not divided into the horrible or the miserable.
0.51	0.90	You will see the beautiful life, not divided into the horrible or the miserable.
0.53	0.91	You will be the beautiful life, not divided into the horrible or the miserable.
0.73	0.97	Happiness is the way to live.
0.79	0.99	Happiness is the way to live. And I'm very happy with the way that I live.
0.83	0.99	My life is wonderful, I'm very happy with the life.
0.84	0.99	We will live in the glorious happiness. And it is really good, it is really good. And I'm very happy with the life that I have.
0.92	0.99	And I'm very happy with the life that I have. And I can't wait to see the next one.

Table 3: Full pareto front of a representative run of sentiment modification for the Woody Allen quote.

to see in the world." From the same run, the most negative sentence on the Pareto front is: "you are the world's worst failure, you have not had good news for the last six months, and you will never find a way to make it up." From the inspirational quote "When the sun is shining I can do anything; no mountain is too high, no trouble too difficult to overcome," it creates a dreary version: "The earth and the mountains beat me hard, the winds blow heavily, the weather is bitter and cold; I cannot do anything."

While the results are not always pleasant, these preliminary experiments highlight that by using a different classification label (or potentially a different model that recognizes different properties of text altogether), it is possible to use LMX for style-transfer of possibly many other styles.

E IMAGE GENERATION EXPERIMENTAL DETAILS

The image generation experiment used Stable Diffusion v1.4 as the text-to-image model for generating images from evolved prompts;

specifically, the 16-bit weight variant (fp16), run from the HuggingFace `diffusers` library.[5] Images were generated at the default 512×512 resolution, and generation was run for 10 diffusion steps per image. While the default number of steps is normally 50, performance was valued over image fidelty. We left the default NSFW filter enabled, which produces a black image when triggered.

Image fitness functions were computed using 8-bit integer RGB images; the maximum fitness is therefore $512 \cdot 512 \cdot 255 = 66{,}846{,}720$, which would be the fitness of a monochromatic image of the target color.

Pythia-deduped 2.8b was used as the LM. This is from the same Pythia model series discussed in Appendix B. Up to 75 tokens were sampled from the LM for each LMX-generated prompt, to stay under Stable Diffusion's limit of 77 tokens in a prompt (Pythia and Stable Diffusion have slightly different tokenizers).

The GA loop used for these experiments was identical to the one from the symbolic regression experiments, but with a smaller population size of 25. The same tournament selection scheme was used, as well as the same 0.05 probability of drawing a new human-written prompt from the initial dataset instead of performing LMX (0.95 probability of generating a new prompt through LMX). Four parents were used as prompts to the LM to produce each LMX-generated child. The number of parents was not tuned for this problem, but chosen based on the results in Figure 2. The parents were given to the language model almost verbatim, with light prompt engineering. Each parent was placed in a paragraph by itself prefixed by "Prompt: ". The list of parents ended with an open "Prompt:" to request that a child be generated.

On an NVIDIA GeForce RTX 3090, with a population size of 25, each generation took about 2 minutes of wall-clock time. The images in Figure 5 each took a little over 3 hours each to evolve over 100 generations. About 75% of the time was spent in the forward pass of the language model, and 25% in text-to-image generation (everything else was negligible).

F PYTHON SODARACERS EXPERIMENTAL DETAILS

Experiments for the Sodaracers domain were carried out using Salesforce's CodeGen suite of language models [56], using the 350M, 2B, and 6B sizes in their 'mono' variant. The 'mono' models were first pre-trained on natural language, before being fine-tuned on a large dataset of code in many languages, before finally being fine-tuned on a dataset of Python only code. All model sampling was done with top p = 0.95, temperature = 0.85, and with a maximum generation length (in addition to the prompt) of 512 tokens. Each experiment for a single combination of seeds, as described in Section 4.5 took around 15 hours to run on a single Nvidia A100 40GB GPU for the 6B model, although for the two smaller model sizes the generation time was much quicker. Use of Nvidia's Triton Inference Server has the potential to speed up sampling from these language models by up to an order of magnitude.

The seven Sodaracers used as our seed programs are described in the appendix of Lehman et al.[46]: the square, radial, wheel, runner, galloper, CPPN-Fixed, and CPPN-Mutable programs (CPPN stands for Compositional Pattern-Producing Network [75]).

[5] https://github.com/huggingface/diffusers

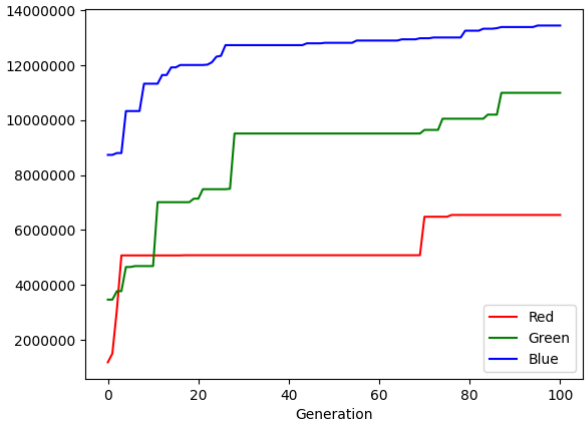


Figure 14: Image generation progress. Max fitness per generation over a single run using each of the three fitness functions.

The Sodaracers were evaluated in a Python simulation of the Sodarace domain [77] written in Box2D (from the Open ELM project [8]). The fitness function was measured as the horizontal distance travelled by an instantiated robot after 1 second of simulation time.

As observed in prior work with few-shot prompting of language models [50], we noticed that success rates (the percentage of generations which resulted in valid Sodaracers) varied dramatically with the order of parent functions in the prompt, sometimes by over 50%. To control for this we averaged our results over every possible permutation of parents.

The main experiments in the paper, described in Section 4.5, prompt the language model with a concatenation of the seed functions, any necessary Python import statements, and the line `def make_walker():` was appended to the end, in order to 'force' the language model to complete a function with this signature.

We also experimented with removing this signature from the end, which produces slightly worse results, particularly in terms of the validation rate. For single-seed prompt mutation, all generations failed to validate, while for LM crossover with two or three parents the validation rate fell by 15% compared with the main prompt.

In addition, we investigated adding an 'instruction' to the end of the LMX prompt, consisting of a string such as 'Combine the starting programs above to create a new program'. This provides some minimal domain customisation to the language model, and is reminiscent of prior work demonstrating that fine-tuning language models on tasks described as instructions can dramatically improve performance on unseen tasks [67, 87].

Our experiments with instruction prompting demonstrate an intriguing direction for future work with instruction-finetuned language models, which may offer improved quality and diversity of evolved programs or strings if prompted in a way compatible with their training data.

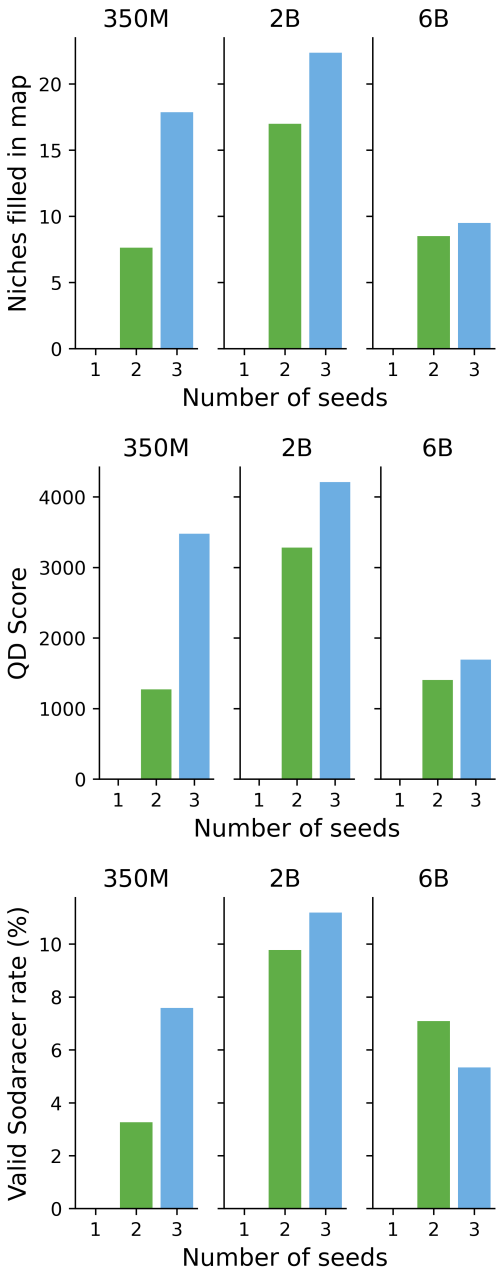


Figure 15: Results from the Sodaracers domain, using a generic domain-free prompt as described in Appendix F, for varying numbers of parents (seeds) in the language model prompt and across language model scale. (top) Number of niches filled in MAP-Elites. (center) Quality-Diversity scores (sum of the fitnesses of all niches in the map) (bottom) Validation rate (%) for the generated Sodaracers. Higher numbers of parents nearly always increases performance in this setting, and the 2B model performs the best.