LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization

Muhammad U. Nasir
umairnasir1@students.wits.ac.za
University of the Witwatersrand
Johannesburg, South Africa

Sam Earle
se2161@nyu.edu
New York University
New York, USA

Christopher W. Cleghorn
christopher.cleghorn@wits.ac.za
University of the Witwatersrand
Johannesburg, South Africa

Steven James
steven.james@wits.ac.za
University of the Witwatersrand
Johannesburg, South Africa

Julian Togelius
julian@togelius.com
New York University
New York, USA

ABSTRACT

Large language models (LLMs) have emerged as powerful tools capable of accomplishing a broad spectrum of tasks. Their abilities span numerous areas, and one area where they have made a significant impact is in the domain of code generation. Here, we propose using the coding abilities of LLMs to introduce meaningful variations to code defining neural networks. Meanwhile, Quality-Diversity (QD) algorithms are known to discover diverse and robust solutions. By merging the code-generating abilities of LLMs with the diversity and robustness of QD solutions, we introduce `LLMatic`, a Neural Architecture Search (NAS) algorithm. While LLMs struggle to conduct NAS directly through prompts, `LLMatic` uses a procedural approach, leveraging QD for prompts and network architecture to create diverse and high-performing networks. We test `LLMatic` on the CIFAR-10 and NAS-bench-201 benchmarks, demonstrating that it can produce competitive networks while evaluating just 2,000 candidates, even without prior knowledge of the benchmark domain or exposure to any previous top-performing models for the benchmark. The open-sourced code is available at https://github.com/umair-nasir14/LLMatic.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; *Lifelong machine learning*; • **Theory of computation → Evolutionary algorithms**.

KEYWORDS

large language models, neural networks, quality-diversity optimization, neural architecture search

ACM Reference Format:
Muhammad U. Nasir, Sam Earle, Christopher W. Cleghorn, Steven James, and Julian Togelius. 2024. LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654017

1 INTRODUCTION

A major challenge in deep learning is designing good neural network architectures. Neural Architecture Search (NAS) is the generic term for various approaches to automating this design process [50]. The idea is to formulate an objective, such as maximum accuracy on a classification problem with a given budget of parameters and training cycles, and cast the problem as a search for the architecture that maximizes the objective. Every test consists of training the candidate network architecture using some form of gradient descent on the chosen benchmark dataset to measure its performance. This typically means that many thousands of architectures are tested and discarded in the process.

Two common algorithmic approaches to NAS are reinforcement learning and evolutionary computation. Reinforcement learning approaches to NAS [20] train a controller (typically another neural network) that outputs network architectures; these network architectures are tested and their performance is used as a reward signal. Evolutionary computation approaches to NAS [25], on the other hand, directly search the space of neural architectures. A population of architectures are kept, and their performance is used as a fitness score. Evolutionary NAS approaches are similar to neuroevolution, which has existed since the 1980s [27, 43], and one might even see NAS as a form of neuroevolution. The main difference is that in NAS, the search process does not concern the parameters of the neural network, only its architecture.

One could argue that search by evolutionary computation or reinforcement learning is quite mindless and wasteful, given how many architectures need to be tested and how uninformed the changes that lead to each new architecture are. Is there some way we can inform the search by exploiting stored knowledge about how to design neural networks? This paper explores the idea that we can do exactly this using code-generating large language models (LLMs). More precisely, we propose combining an LLM with an evolutionary algorithm to generate new architectures that have high network architectural diversity and state-of-the-art performance.

The argument for this is simply that modern LLMs fine-tuned on code are very capable [39]. Given the amount of machine learning

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07...$15.00
https://doi.org/10.1145/3638529.3654017

code they have been trained on, it is not surprising that they can design good neural network architectures. However, an LLM by itself cannot, in general, find an optimal architecture for a given problem, as it cannot test architectures and learn from its experiments. Therefore, we propose combining the domain knowledge of code-generating LLMs with a robust search mechanism.

While generating a single architecture that maximizes a given objective is useful for many cases, there is often more value to generating a set of architectures that vary across some relevant dimensions. For example, one might want to have a set of architectures that vary in their parameter counts or depths. This helps in understanding the trade-offs between various desirable metrics and could assist in making better-informed decisions about which architecture to use for a specific application. For example, one might want a range of networks for edge deployments to clients with different RAM sizes. To enable this, the solution we propose here leverages quality-diversity search [36], specifically a version of the MAP-Elites algorithm [28].

Our main contribution is a novel LLM-based NAS algorithm, `LLMatic`, that utilizes the power of two QD archives to search for competitive networks with just $2,000$ evaluations. We empirically show the performance of `LLMatic` on the CIFAR-10 dataset and the NAS-bench-201 benchmark where `LLMatic` searches for networks with performance similar to state-of-the-art results.

2 RELATED WORK

Designing neural architectures can be an expensive and unintuitive process for human designers. Neural Architecture Search (NAS) aims to automatically find architectures capable of strong performance after training [14]. Bayesian methods are a popular choice given their low sample complexity and the fact that evaluating each architecture (by training it) can be computationally expensive [21]. Alternatively, reinforcement learning can be used to train an agent (usually another neural network) to output candidate architectures for a given task, with the performance after training of the candidate architecture acting as a reward signal [20]. Evolutionary methods can also be used to search directly through the space of possible architectures [25]. Similarly, Monte Carlo Tree Search has also been used to search [51]. In all cases, a human designer must manually define a set of atomic network components or edit actions for use in network search/generation.

To avoid having the designer constrain the space of possible architectures prior to search, we turn to code-generating large language models (LLMs)—large models trained auto-regressively on massive datasets of code (e.g. public repositories hosted on Github). These LLMs are based on the transformer architecture [48] that has obtained state-of-the-art performance in natural language modelling [1, 5, 32, 37]. They have also been successfully used in specific applications, such as for video game level design [7, 33, 44] or code generation [39].

Recently, LLMs have been used for evolving code by framing code-generation as an evolutionary problem. Evolution through Large Models (ELM) [23] casts LLMs as evolutionary operators within a MAP-Elites [28] algorithm tasked with evolving a robot's morphology at code-level. EvoPrompting [6] is an LLM-based method that is somewhat similar to ours in that it uses code-LLMs as mutation and crossover operators to perform NAS. It is tested on the MNIST-1D classification task [17] and the CLRS algorithmic reasoning benchmark [49]. Since performance can generally be trivially increased by simply adding parameters to the model, an additional penalty is added to the fitness of a candidate neural architecture corresponding to its model size. This incentivizes the discovery of small models with effective architectures. In our method, we instead consider model complexity (in terms of FLOPS) as a diversity metric, searching for high-performing models of a variety of sizes. GENIUS [53] is another LLM-based NAS algorithm that uses GPT-4 to simply search through straight-forward prompting.

Quality Diversity (QD) methods [36] are a family of evolutionary algorithms that, in addition to optimizing a fitness metric, search for a diversity of individuals according to some user-specified "behavioral descriptors". Instead of keeping a population of the fittest individuals, QD methods such as MAP-Elites [28] maintain an "archive" of individuals, where this archive is partitioned into cells, with each cell corresponding to individuals exhibiting a particular range of values along each behavioral descriptor.

QD methods are valuable in domains such as robot control, where it is useful to learn diverse high-quality trajectories, in case one solution should become unavailable during deployment because of a physical obstruction or mechanical malfunction [9]. Another motivating factor is that greedily searching for the fittest individual may not be desirable in deceptive domains. Here, maintaining a diversity of fit individuals may protect the population from falling into local optima [15]. Conversely, diverse, unorthodox solutions may provide valuable "stepping stones" on the path to globally fit individuals.

3 APPROACH

`LLMatic` begins its search with a very basic neural network, inspired by the work of [40] which suggests that neuroevolution tends to perform better when starting with a small network. In `LLMatic`, we use a novel dual-archive cooperative QD optimization approach, in which two separate archives are used to store complementary components that can be combined to solve a given task. The first archive stores neural networks, where the width-to-depth ratio and Floating Point Operations per Second (FLOPS) of a network are the behavioural descriptors. The width-to-depth ratio is a division of the width and the depth of the network. To specify the width, we use the maximum of the output features of all layers, while depth is simply the number of layers. Note that we choose FLOPS instead of parameter count because FLOPS correlates better with actual time spent training a network [2]. We call this archive the "network archive". The fitness function for the networks in this archive is defined as the test accuracy of the network after training. The second archive, called the "prompt archive", contains the prompt and temperature used for generating code, which are also the behavioural descriptors. The *temperature* of an LLM is a hyperparameter that controls the degree of stochasticity in the selection of output tokens: a lower temperature means that the LLM will select tokens more deterministically, while a higher value will result in more diverse output. The selection of prompt and temperature depends on a curiosity score [10], governed by whether the

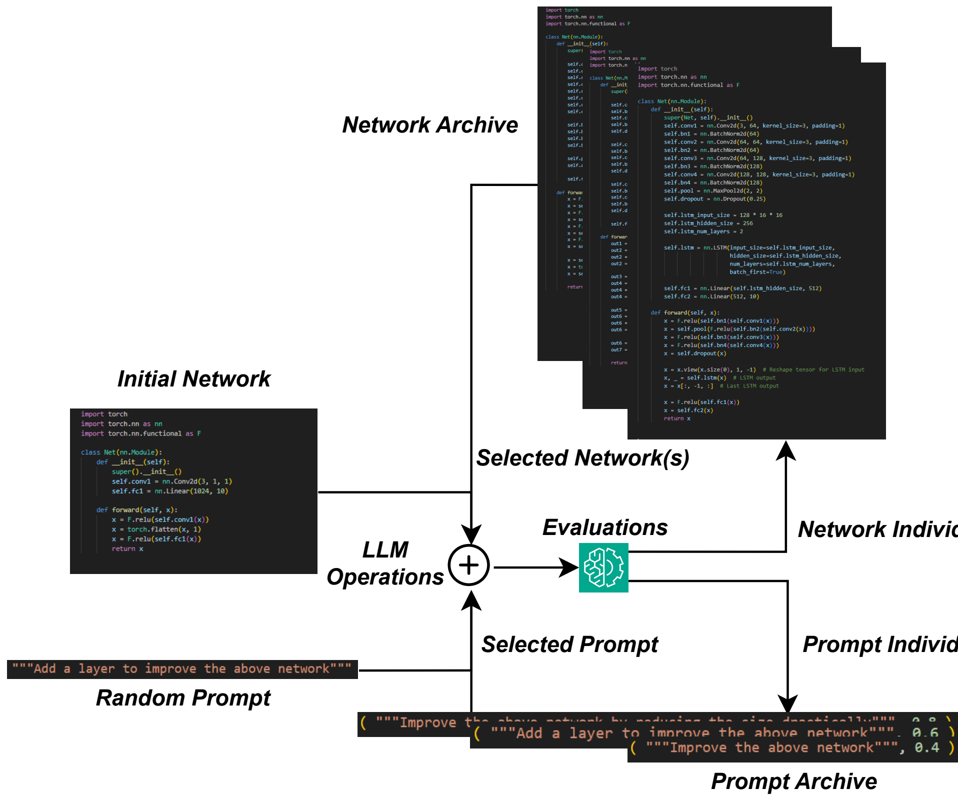


Figure 1: Illustrated in the figure is the flow of LLMatic. In the initial round of evolution, an initial network with a random prompt goes through a mutation operation. Network individual and prompt individual are then evaluated to be stored in separate archives. During the evolutionary loop, the selected prompt and network go through an evolutionary operation (the prompt is fixed if the operation is crossover) to create more networks and prompt individuals to fill and illuminate the archives.

generated network was added to the network archive. The fitness of prompt individuals depends on whether the network was better than the previous generation's best score. Figure 1 illustrates the flowchart of the approach LLMatic uses, while Algorithm 1 shows the complete search process of `LLMatic` in pseudocode.

In the first generation, a simple neural network with one convolutional and one fully connected layer initiates the evolution (line 1 of Algorithm 1). A prompt is selected at random to generate an initial batch of networks (lines 5–6 of Algorithm 1). These networks are evaluated and an attempt is made to add them to the network archive as a random initialization for MAP-Elites. Concurrently, we mutate the temperature based on the fitness of the network, increasing it if the fitness increases and vice versa (lines 21–23 of Algorithm 1). An increase in temperature is desirable if we want the LLM to explore, while decreasing the temperature will result in the LLM exploiting in an attempt to achieve better fitness than before. Once we calculate the fitness of the prompt individual, we add the score to a collective prompt fitness score, after which we try to populate the prompt archive. The collective prompt fitness score determines the overall fitness of each individual in the prompt archive as it gives each prompt a fitness score.

Once either of the archives reaches a specified capacity, we introduce neural network training and evolutionary operators in the process (lines 7–20 of Algorithm 1). With a certain probability at each generation, a decision is made on whether to perform crossover or mutation to produce N new offspring. If the crossover operator is chosen, we select N random network individuals, locate their closest networks in the archive, and carry out a crossover operation instructed by a prompt (lines 15–16 of Algorithm 1). No individual is added to the prompt archive when a crossover is performed. If the mutation operation is selected, we pick the most curious prompt individual and a random network individual. For exploration, we also select random prompts. In both cases, each network is trained for a certain number of epochs and an attempt is made to add the network to the archive. Likewise, a prompt individual is added as previously described. This process continues for a predetermined number of generations. Refer to the supplementary material for pseudocode on mutation operators, crossover operators, temperature mutation and addition to archives.

4 EVALUATING LLMATIC

To evaluate `LLMatic`, we use CIFAR-10 [22], a commonly used dataset for NAS [42, 52]. We perform extensive ablation studies to demonstrate that `LLMatic` benefits from each of its components

Algorithm 1: LLMatic

```
Initialize network and prompt archives.
while number of generations < maximum generations
    foreach network in batch of networks do
        if number of individuals in archives < set threshold then
            Mutate the initial network with a random prompt.
            Get the network and prompt individuals and add them to the respective archives.
        else
            Randomly choose mutation or crossover as the evolutionary operator.
            if evolutionary operator == mutation then
                Select network and prompt.
                Mutate the selected network with the selected prompt.
                Train or query the generated network.
                Get the network and prompt individuals and store them.
            else
                Selection of networks for crossover.
                Perform crossover on selected networks with a fixed prompt.
                Train or query the generated network.
                Store the generated network individual.
            end
        end
    end
    Evaluate all networks to find the test accuracies.
    Mutate the temperature for the next generation.
    Add the batch network individuals and the corresponding prompt individuals in the respective archives.
end
```

during search. Once our algorithm is validated, we extend our experiments to *NAS-bench-201*. The *NAS-bench-201* benchmark [26] is a dataset enumerating all possible neural architectures within a given search space (i.e. of a fixed number of nodes/layers and edges/operations) and the corresponding test accuracy of each architecture after training on a given dataset (e.g. CIFAR-10), allowing researchers to explore the tradeoffs of various NAS algorithms without needing to retrain each candidate network during search.

4.1 Setting up LLMatic

Dataset: The CIFAR-10 dataset is made up of $60,000$ color images, each with a resolution of 32×32 pixels, and divided into 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The dataset is partitioned into five groups for training and one group for testing, each group holding 10,000 images. Each test group consists of an exact count of 1,000 images from each category selected randomly. The training groups hold the remaining images, which are arranged in random order. As a result, some training groups might contain more images from one category compared to others. Nonetheless, collectively, the training groups have an exact total of 5,000 images from each category.

Initial Neural Network: `LLMatic` starts off with a simple neural network with one convolutional layer that takes in 3 input channels, with 1×1 kernel size and 1 output channel which connects to a dense layer of size 1024. These hidden neurons are connected via another dense layer to 10 output neurons (as we have 10 classes). Rectified Linear Unit (ReLU) [30] is the activation function used in all layers. All of our networks are generated in PyTorch [35].

Generating Neural Networks: At each generation, we generate a batch of 100 new offspring. Each network generated is trained for 50 epochs. The networks are optimized by stochastic gradient descent [4] with the learning rate set to 0.001 and momentum set at 0.9 for all networks. We use cross entropy loss as our measure for the fitness of the trained network.

For evolutionary operators, we set a probability of 0.7 for mutation and 0.3 for crossover as after experimentation, we found that mutation creates consistently more trainable neural networks. We initialize the temperature parameter (used when sampling the code-generating LLM) to 0.6. For temperature mutation, half of the population is generated by the prompt individual temperature mutated uniformly at random between -0.1 to 0.1. The other half is generated by the temperature obtained from the prompt individual itself. If the fitness of the generated network is better than or equal to the best fitness of the previous generation, we increase the temperature by 0.05 and if it is worse than the best fitness of the previous generation, we decrease it by 0.05. For the crossover operator, we select 10 random networks and find their 2 or 3 nearest neighbours, based on the distance of niches, in the network archive to perform crossover. We set the LLM temperature to be 0.7 for network generation.

Quality Diversity Optimization: For our QD optimization algorithm, we choose a variant of MAP-Elites—Centroidal Voronoi Tessellation (CVT-MAP-Elites) [47]—which can be seen as a generalization of MAP-Elites that is intended to scale MAP-Elites to high-dimensional behavior spaces; CVT-MAP-Elites outperforms standard MAP-Elites in high-dimensional spaces while matching performance lower-dimensional scenarios such as those studied in the present work.

CVT-MAP-Elites automates the sub-division of the archive by identifying k cell centroid locations exhibiting an even spread through the behavioural descriptors space. Here, k corresponds to the total number of evolutionary "niches" in the archive. We use the *pymap_elites*[1] implementation for our experimentation. We use a k-d tree [3] to create and write centroids to the archive and find the nearest neighbors using a Euclidean distance metric [12].

Each of our QD archives has 2 dimensions (behavioral descriptors), with 100 niches spread across them. We set the number of random initial networks to 10. For the network archive, we have the width-to-depth ratio of the network as our first dimension and the FLOPS of the network as the second dimension. The width-to-depth ratio has a lower limit of 0 and an upper limit of 200. The

[1] https://github.com/resibots/pymap_elites

minimum FLOPS is set to 200 MegaFLOPS and the maximum is set to 5 GigaFLOPS. This range is set after experimentation.

For the prompt archive, we have the prompt encoded as an integer as the first dimension and temperature as the second dimension. The maximum value of the prompt is 16, the number of prompts used in the system. The maximum temperature value is set to 1 as it can never increase beyond that for our LLM. The lower limit for all dimensions is 0.

For the network archive, we simply select a random network while for the prompt archive, we select the most curious prompt individual, which depends on the curiosity score. This curiosity score is incremented by 1.0 if the selected prompt adds the generated network to the network archive, decreased by 0.5 if the network is not added, and reduced by 1.0 if the created network is untrainable. If the generated network has better fitness than the previous generation's best network, the collective prompt fitness score for the prompt in the prompt individual is increased by 1; otherwise, it is unchanged. We use prompts that are generalizable to any problem in any domain. Refer to supplementary material for an example of mutation and crossover prompts.

Code Generating LLM: We use the pre-trained CodeGen [34] LLM to generate neural networks. CodeGen is an autoregressive decoder-only transformer with left-to-right causal masking. CodeGen is first trained on `ThePile` dataset with random initialization and is called CodeGen-NL. CodeGen-Multi is initialized with CodeGen-NL and is trained on `BigQuery` dataset. Lastly, CodeGen-Mono is initialized with CodeGen-Multi and is trained on `BigPython`. CodeGen is trained to be in various parameter sizes, but we use 6.1 Billion parameter variant of CodeGen-Mono due to computational constraints.

`ThePile` dataset [16] is an 825.18 GB English text corpus. CodeGen selects a subset of the Google `BigQuery` dataset which contains 6 programming languages, namely C, C++, Go, Java, JavaScript, and Python. The authors collected a large amount of permissively licensed Python code from GitHub in October 2021, and named it `BigPython`. The size of `BigPython` is 217.3 GB.

CodeGen-6B has 33 layers and 16 heads with 256 dimensions per head. The context length is 2048 and the batch size is 2 million tokens. Weight decay is set to 0.1. $0.4e^{-4}$ is the learning rate. Warmup steps are set to $3k$ while total steps for training are 150k.

4.2 Ablation Study

As we have many components in `LLMatic`, we choose to do a thorough ablation study to determine the effect of each component on overall performance. The following are the components tested for the ablation study:

- `Network-Archive-LLMatic`: `LLMatic` with only the network archive. To achieve this, we create a population of prompt individuals. The population is fixed to 100 individuals initialized with random individuals. We have only one fitness score for this population, which is calculated as +1 if a network is added in the network archive, -0.5 if the network is not added and -1 if the network is not trainable. After we generate the network, we mutate the temperature by adding 0.1 if the network is added in the network archive and -0.1 if the network is not added.
- `Prompt-Archive-LLMatic`: `LLMatic` with only the prompt archive. To achieve this, we create a population of networks. The fitness function for the population of networks is accuracy. We keep the population to 100 individuals. With a similar probability as `LLMatic`, we select mutation or crossover operator. For the crossover operator, we select the individual that is closest to the structure of the selected network. For network similarity, we use cosine similarity and we choose the networks with higher scores. For the mutation operator, similar to `LLMatic` we mutate half of the networks from the most curious prompt individuals and half from random individuals.
- `Mutation-Only-LLMatic`: `LLMatic` using only mutation.
- `Crossover-Only-LLMatic`: `LLMatic` using only crossover.
- `Random-NN-Generation`: Neural network generation without evolution. We generate 100 networks per generation for 20 generations as a fair comparison to LLMatic, which generates the same number per batch. We apply the prompt "Create a neural network that inherits from nn.Module and performs better than the above neural network" and we add the initial network with this prompt.

4.2.1 Ablation Results and Discussion. In this section, we will discuss the results of the experiments that we set up in the previous section. We first discuss the best accuracy per generation, illustrated in Figure 2. This will lead our discussion to trainable networks generated by changing the crossover and mutation probabilities (Figure 4). Then we will discuss how archives are illuminated Figure 3. Some of the generated networks are shown in the supplementary material.

Figure 2 illustrates that each component of `LLMatic` is necessary. `Mutation-Only-LLMatic` and `Network-Archive-LLMatic` are the closest to `LLMatic`, which validates our choice to weight the probability of mutation higher. `Crossover-Only-LLMatic` performs the worst, as it does not benefit from the exploration abilities provided by the mutation operator [46]. Both operators (mutation and crossover) together provide exploration and exploitation abilities to `LLMatic`, which appear necessary to find high-quality and diverse networks. `Prompt-Archive-LLMatic` performs poorly, indicating that the network archive is an important aspect in finding high-performing networks. However, both archives together demonstrate competitive results.

We use EfficientNet-B0, which is the state-of-the-art network on CIFAR-10 Tan and Le [42] as an indicator of where our algorithm stands. EfficientNet-B0 was searched via methods applied by Tan et al. [41] and is slightly larger than the original study as they were targeting more FLOPS. The original study required 8,000 evaluations, while `LLMatic` requires 2,000 evaluations to find a competitive network. EfficientNet-B0 was first trained on the ImageNet dataset [11] and then on CIFAR-10 via transfer learning [45]. This is an advantage for EfficientNet-B0 as ImageNet has many classes and is an order of magnitude larger dataset.

Figure 3 demonstrates how each archive is filled on average. We can see that the prompt archive contains high-performing individuals who have the first few prompts and higher temperatures. Some of the high-performing individuals do have lower temperatures,

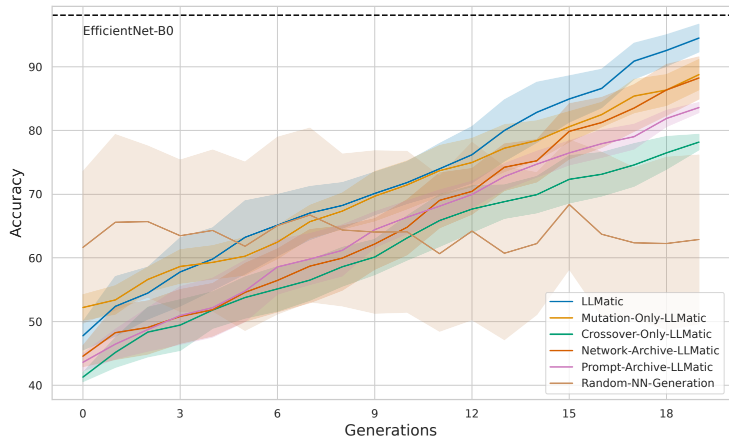
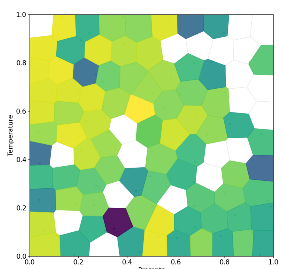
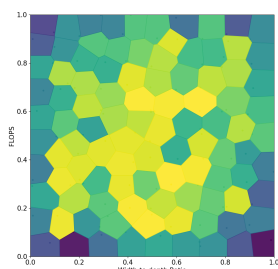


Figure 2: The illustration of the best accuracy per generation for `LLMatic` and all ablation studies. Each experiment is conducted with 30 seeds. The shaded region is the standard deviation while the solid line represents the mean. EfficientNet-B0 is the best-performing EfficientNet on CIFAR-10.



(a) Prompt archive: Prompts encoded as integers on the x-axis, normalised to be in range 0-1 for CVT-MAP-Elites as all points are within 0–1. On the y-axis, we have the temperature that controls LLMs exploration ability. As 1 is the maximum temperature, there is no need for normalisation.



(b) Network archive: Width-to-depth ratio on the x-axis. The range for the Width-To-Depth ratio is from 0–200 normalised to 0–1. On the y-axis, we have Floating Point Operations per Second (FLOPS). We have a range of 200 Mega FLOPS to 5Giga FLOPS. This range is normalised to 0-1 for CVT-MAP-Elites.

Figure 3: An illustration of archives generated by `LLMatic`. We have selected the archive with the median number of cells filled in experiments over 30 seeds. Figure 3a shows the prompt archive, while Figure 3b shows the network archive. The lighter the colour of the filled cell, the better fitness of the individual. White indicates that the cell is empty.

which suggests that sometimes it is useful to generate neural network layers in a less stochastic manner. For network archives, we observe a diversity of high-performing networks with respect to both FLOPS and width-to-depth ratio. More than 20 individuals are competitive networks in this archive.

To investigate our choice of probabilities for crossover and mutation (0.3 and 0.7, respectively), we observe the number of trainable networks generated per generation (see Figure 4). We use this as a measure, since the more functional individuals we have, the greater the chance of high-performing individuals. For this purpose, we train `LLMatic` with uniform probabilities, and 0.3 for mutation and 0.7 for crossover. We observe that uniform probabilities are still competitive with the original setting, while increasing the crossover probability makes it worse. The results of these experiments and results of the ablation study for `Crossover-Only-LLMatic` and `Mutation-Only-LLMatic` lead us to the conclusion that mutation should be given more probability of being selected.

5 EXPERIMENTS ON NAS-BENCH-201

Next, we extend our experimentation of `LLMatic` to the NAS-bench-201 benchmark [13], which searches a cell block for a constant neural network structure. The structure is initiated with one 3×3 convolution with 16 output channels and a batch normalization layer [19]. The main body of the skeleton includes three stacks of cells, connected by a residual block. Each cell is stacked 5 times, with the number of output channels as 16, 32 and 64 for the first, second and third stages, respectively. The intermediate residual block is the basic residual block with a stride of 2 [18], which serves to downsample the spatial size and double the channels of an input feature map. The shortcut path in this residual block consists of a 2×2 average pooling layer with a stride of 2 and a 1×1 convolution. The skeleton ends with a global average pooling layer to flatten the feature map into a feature vector. Classification uses a fully connected layer with a softmax layer to transform the feature vector into the final prediction.

The specified cell within the search domain is depicted as a densely connected directed acyclic graph with four nodes and six

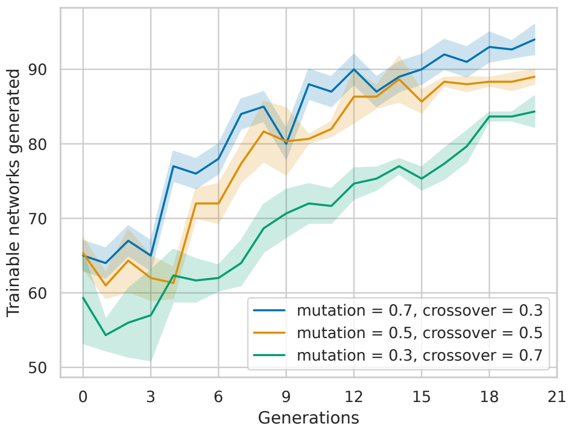


Figure 4: The illustration of how many trainable networks are created in a generation. The total number of networks created is 100 per generation. This illustration is calculated over 10 runs. The shaded region is the standard deviation.

edges; here, nodes symbolise feature maps while edges denote operations. There are five possible operations: (1) zeroize, (2) skip connection, (3) 1×1 convolution, (4) 3×3 convolution, and (5) 3×3 average pooling layer. Zeroize drops out the associated edge operation. Given five operations to choose from, the aggregate count of potential search spaces is $5^6 = 15625$ cell combinations. Evaluations are carried out on CIFAR10, CIFAR100 [22], and ImageNet16-120 [8]. ImageNet16-120 is a variant of ImageNet dataset [38] which is down-sampled to 16x16 image sizes and contains the first 120 classes.

5.1 Results

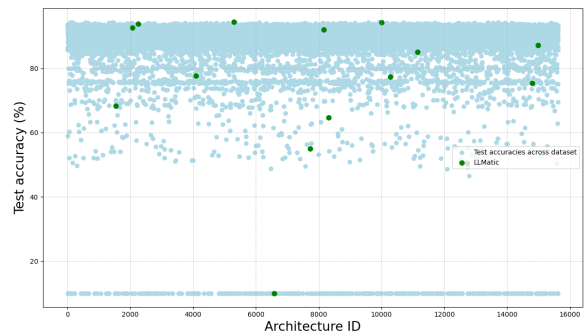
To remain consistent with our previous experiments, `LLMatic` searches for 20 generations and 100 cells in a generation. We curate the prompt to cater for a controllable generation by restricting it to the five operations. Refer to supplementary material for an example of how we generate queryable cells. For our network archive, we take minimum and maximum FLOPS as the bounds for the behaviour descriptor.

Table 1: A comparison of test accuracy on the NAS-bench-201 benchmark. We provide the optimal accuracy for reference, which is the maximum accuracy that can be achieved in NAS-bench-201. The results for `LLMatic` are averaged over 10 runs.

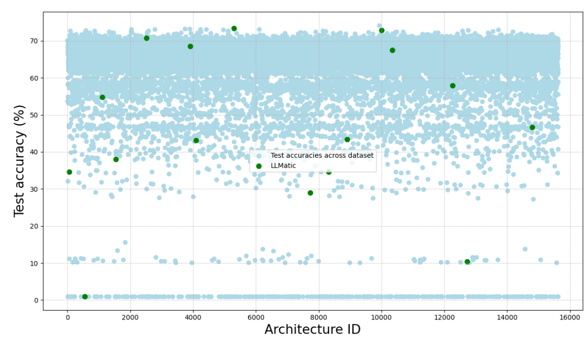
Method	CIFAR-10	CIFAR-100	ImageNet16-120
DARTS	54.30±0.00	15.61±0.00	16.32±0.00
Random Search	93.70±0.36	71.04±1.07	44.57±1.25
GENIUS	93.79±0.09	70.91±0.72	44.96±1.02
Λ-DARTS	94.36±0.00	73.51±0.00	46.34±0.00
LLMatic	94.26±0.13	71.62±1.73	45.87±0.96
Optimal	94.47	74.17	47.33

We compare our results with the GPT-4-based NAS algorithm GENIUS [53], which serves as an LLM baseline, as well as random search. We also compare to prior work, including DARTS [24] and Λ-DARTS [29], which achieves near-optimal results. As Table 1 indicates, `LLMatic` outperforms the GPT-4-based NAS, and produces results that are near state-of-the-art, although it has more variation than the competing methods.

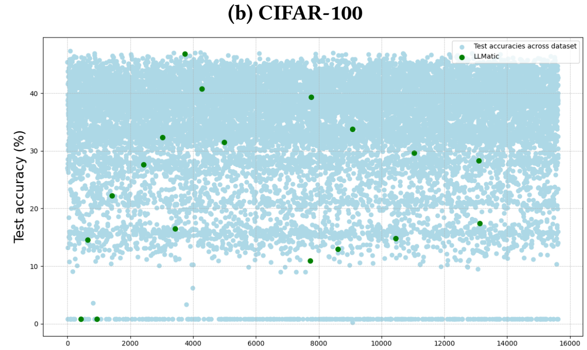
Furthermore, in Figure 5 we investigate the networks discovered by `LLMatic` over each generation. We observe that the distribution of found networks is spread wide in the search space. This is due to the procedural nature and exploration capabilities of `LLMatic`



(a) CIFAR-10



(b) CIFAR-100



(c) ImageNet16-120

Figure 5: Illustration of test accuracies of all networks across all datasets and best-found networks in each generation by `LLMatic`.

Table 2: Maximum rank achieved by LLMatic on each dataset in NAS-bench-201.

Method	Rank
CIFAR-10	2
CIFAR-100	2
ImageNet16-120	11

through the prompt archive. To demonstrate near-to-optimal networks we illustrate in Table 2 the maximum ranked networks based on test accuracies searched by LLMatic.

6 CONCLUSION AND FUTURE WORK

To conclude, we present LLMatic: a novel neural architecture search (NAS) algorithm that harnesses the power of large language models (LLMs) and Quality-Diversity (QD) optimization algorithms. LLMatic successfully finds competitive networks that are diverse in architecture. We show empirically that LLMatic can find more than *20* competitive networks in CIFAR-10 and near-to-optimal networks in NAS-bench-201, using only *2000* evaluations. LLMatic decreases the max population size per generation to only *100*. LLMatic achieves this while relying on a *6.1B* parameter language model. Furthermore, we show that each component in LLMatic is necessary. We conducted an extensive ablation study and found that LLMatic finds the network with the best accuracy among other variants.

LLMatic achieves this with many constraints in hand. Firstly, we use CodeGen-6.1B code generation LLM, which is a smaller language model when compared to existing LLMs. This demonstrates the computationally efficiency of LLMatic, and gives us reason to believe that further gains could be unlocked by incorporating larger language models Secondly, due to computational resources, we keep our searches to 2000, and still find competitive networks.

In future work, LLMatic should be compared to other NAS methods on other computer vision and natural language processing tasks. As neuroevolution is similar to NAS, LLMatic could be compared to reinforcement learning benchmarks as well. With this, LLMatic can be used in tasks such as open-ended learning as well [31].

7 ACKNOWLEDGMENTS

The authors acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources to this research project.

REFERENCES

[1] David Ifeoluwa Adelani, Jesujoba Oluwadara Alabi, Angela Fan, Julia Kreutzer, Xiaoyu Shen, Machel Reid, Dana Ruiter, Dietrich Klakow, Peter Nabende, Ernie Chang, et al. 2022. A few thousand translations go a long way! leveraging pre-trained models for african news translation. *arXiv preprint arXiv:2205.02022* (2022).

[2] Helon Vicente Hultmann Ayala, Daniel M Muñoz, Carlos H Llanos, and Leandro dos Santos Coelho. 2017. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. *Control Engineering Practice* 60 (2017), 124–132.

[3] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[4] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, 177–186.

[5] 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.

[6] Angelica Chen, David M Dohan, and David R So. 2023. EvoPrompting: Language Models for Code-Level Neural Architecture Search. *arXiv preprint arXiv:2302.14838* (2023).

[7] 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).

[8] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819* (2017).

[9] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. Robots that can adapt like animals. *Nature* 521, 7553 (2015), 503–507.

[10] Antoine Cully and Yiannis Demiris. 2017. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 245–259.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[12] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. 2015. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine* 32, 6 (2015), 12–30.

[13] Xuanyi Dong and Yi Yang. 2020. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326* (2020).

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[15] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. 2019. Are quality diversity algorithms better at generating stepping stones than objective-based search?. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 115–116.

[16] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027* (2020).

[17] Sam Greydanus. 2020. Scaling down deep learning. *arXiv preprint arXiv:2011.14439* (2020).

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[19] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[20] Yesmina Jaafra, Jean Luc Laurent, Aline Deruyver, and Mohamed Saber Naceur. 2019. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing* 89 (2019), 57–66.

[21] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems* 31 (2018).

[22] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[23] 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).

[24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[25] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. 2021. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems* (2021).

[26] Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. 2022. NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy. *arXiv preprint arXiv:2201.13396* (2022).

[27] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. 1989. Designing Neural Networks Using Genetic Algorithms.. In *ICGA*, Vol. 89. 379–384.

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

[29] Sajad Movahedi, Melika Adabinejad, Ayyoob Imani, Arezou Keshavarz, Mostafa Dehghani, Azadeh Shakery, and Babak N Araabi. 2022. Λ-DARTS: Mitigating Performance Collapse by Harmonizing Operation Selection among Cells. *arXiv preprint arXiv:2210.07998* (2022).

[30] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference*

on machine learning (ICML-10). 807–814.
[31] Muhammad Umair Nasir, Michael Beukman, Steven James, and Christopher Wesley Cleghorn. 2022. Augmentative Topology Agents For Open-ended Learning. *arXiv preprint arXiv:2210.11442* (2022).
[32] Muhammad Umair Nasir and Innocent Amos Mchechesi. 2022. Geographical distance is the new hyperparameter: A case study of finding the optimal pre-trained language for English-isiZulu machine translation. *arXiv preprint arXiv:2205.08621* (2022).
[33] Muhammad U Nasir and Julian Togelius. 2023. Practical PCG Through Large Language Models. *arXiv preprint arXiv:2305.18243* (2023).
[34] 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).
[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[36] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. 2016. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* (2016), 40.
[37] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.
[39] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.
[40] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.
[41] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2820–2828.
[42] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.
[43] Manoel Tenorio and Wei-Tsih Lee. 1988. Self organizing neural networks for the identification problem. *Advances in Neural Information Processing Systems* 1 (1988).
[44] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–8.
[45] Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 242–264.
[46] Sami Ullah, Abdus Salam, and Mohsin Masood. 2022. Analysis and comparison of a proposed mutation operator and its effects on the performance of genetic algorithm. *Indonesian Journal of Electrical Engineering and Computer Science* 25, 2 (2022), 1208–12168.
[47] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 2017. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation* 22, 4 (2017), 623–630.
[48] 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).
[49] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. 2022. The CLRS algorithmic reasoning benchmark. In *International Conference on Machine Learning*. PMLR, 22084–22102.
[50] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. 2023. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727* (2023).
[51] Martin Wistuba. 2017. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420* (2017).
[52] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*. PMLR, 7105–7114.
[53] Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. 2023. Can GPT-4 Perform Neural Architecture Search? *arXiv preprint arXiv:2304.10970* (2023).