scientific reports

Check for updates

OPEN

An automated multi parameter neural architecture discovery framework using ChatGPT in the backend

Md Hafizur Rahman✉, Zafaryab Haider & Prabuddha Chakraborty

Building efficient neural network architectures for a given dataset can be a time-consuming task requiring extensive expert knowledge. This task becomes particularly challenging for edge artificial intelligence (AI) because one has to consider additional parameters such as power consumption during inferencing, model size, and inferencing speed. In this article, we introduce a novel framework designed to automatically discover new neural network architectures based on user-defined parameters, an expert system, and an LLM trained on a large amount of open-domain knowledge. The proposed framework (LEMONADE) can be easily used by non-AI experts, does not require a predetermined neural architecture search space, and considers a large set of edge AI parameters. We implement and validate this proposed neural architecture discovery framework using CIFAR-10, CIFAR-100, ImageNet16-120, EuroSAT, Malaria Parasite, and IMDb datasets while primarily using ChatGPT-4o as the LLM component. We have also explored the possibilities of using Gemini-Pro as the LLM component. Neural networks generated using LEMONADE for CIFAR-10 (95.54% test accuracy) and CIFAR-100 (79.43% test accuracy) demonstrated state-of-the-art performance in terms of final model accuracy. We have also observed near state-of-the-art performance (in terms of accuracy) for the ImageNet16-120 dataset. Moreover LEMONADE was able to generate effective neural networks, satisfying different edge AI requirements across additional datasets such as EuroSAT.

Neural networks (NN) have found extensive application across various fields such as healthcare[1–3], surveillance[4,5], Industry 4.0[6–8], and Internet of Things (IoT)[9–11]. A neural network can be composed of a large number of layers of different types while sporting diverse hyperparameters. Hence, for a given dataset/application: (1) finding the right set of neural layers; (2) connecting them in the right topology; and (3) selecting the most optimal hyperparameters for each layer can be a daunting task requiring a large amount of computation resources, human expert involvement, and time. Requiring a given neural network to perform (during inferencing) under specific resource-constrained conditions (a case for many IoT/Edge devices) can add to the complexity of the neural architecture search process. For example, designing a neural network to have more than $x\%$ accuracy for a given task is a hard problem to solve but it becomes harder if we further constrain the problem with additional parameters such as frames-per-second (FPS) requirements during inferences and power consumption limits.

Traditional neural architecture search (NAS) frameworks are typically designed to identify the best architecture within a specified search space. This approach is constrained by its pre-defined search space, which limits its capacity for generating novel neural network architecture (outside the search space). Additionally, most NAS frameworks prioritizes final model accuracy leading to very high search-cost (time, energy consumption) and poor edge AI performance (low FPS and high inferencing energy).

To mitigate these concerns we propose a large language model guided neural architecture discovery (LEMONADE) framework that can allow the discovery of novel neural network architecture without relying on a pre-defined search space. This framework is designed to allow the network-builder to efficient trade-off between: (1) Final model accuracy; (2) Neural search/discovery speed and energy consumption; (3) Final model energy consumptions and inferencing frames per second. These objectives are enforced through an iterative approach utilizing a large language model (LLM) and an expert system for driving the LLM towards the target discovery. The expert system will use a set of configurable rules and several user-defined metrics to generate a set of instructions for the LLM leading to progressive refinement of the generated neural architecture.

Department of Electrical and Computer Engineering, University of Maine, Orono, ME 04469, USA. ✉email: md.hafizur.rahman@maine.edu

To validate the LEMONADE framework, we perform extensive experimentation using the CIFAR-10, CIFAR-100, ImageNet16-120, EuroSAT, Malaria Parasite, and IMDb datasets. We use the framework to generate different neural networks for diverse application requirements and priorities. Neural networks generated using LEMONADE for CIFAR-10 (95.54% test accuracy) and CIFAR-100 (79.43% test accuracy) demonstrated state-of-the-art performance in terms of final mode accuracy. For ImageNet16-120 LEMONADE was also able to generate fairly competitive architectures (42.95% test accuracy). LEMONADE is also very efficient in terms of model generation and training, demonstrating notable reduction in network search/discovery time and associated energy consumption. While using GPT-4o[12], as the backend LLM, LEMONADE was able to generate and train CIFAR-10 models in about 5.8 hours consuming only about 1.20 kWh-PUE energy. LEMONADE is also capable of prioritizing metrics beyond accuracy, which enables the creation of neural architectures that are optimal for different IoT/Edge requirements such as high speed inferencing at low power. This is achieved through efficiently trading-off model accuracy as demonstrated by our experimental results across several datasets/applications. LEMONADE has generated novel neural architectures from scratch, thereby, paving a new opportunity for search-space agnostic neural architecture search research. We have also validated the framework while using Gemini-Pro as the LLM component. To summarize, we:

1. Formalize and design a cost-effective and search-space agnostic neural architecture discovery framework (LEMONADE) leveraging LLMs.
2. Formulate an expert system with associated rules and relevant metrics that is capable of driving a given LLM toward discovering different neural architectures.
3. Implement LEMONADE as a highly configurable/efficient tool for immediate application and easy future extensions.
4. Qualitatively and quantitatively evaluate LEMONADE using CIFAR-10, CIFAR-100, ImageNet16-120, EuroSAT, Malaria Parasite, and IMDb datasets for diverse settings and application requirements.

Background and motivation

Next, we will briefly describe relevant related works and discuss the motivations that drove the development of LEMONADE.

Neural architecture search

Methods of neural architecture search (NAS) are extensively used across various applications such as image processing[13–16], signal processing[17–19], object detection[20,21], and natural language processing[22,23]. It involves identifying the best neural network for a given task through repeated trials, traditionally judged solely based on final model accuracy. The early NAS techniques worked mainly based on the evolutionary algorithms (EA)[24] and reinforcement learning (RL)[25]. Although these methods showed promising results in building quality NN, they require substantial computing power and time. To solve this issue, weight-reusing[26] approaches were proposed that avoid the necessity of training each design from the beginning, resulting in low computation costs. One-shot approaches for NAS[27] were also proposed which involves training a large network called SuperNet that incorporates every conceivable architecture within the search domain. Differentiable Neural architecture search (DNAS)[28] is another weight re-using approach where all the SubNet parameters are optimized by gradient descent.

Most NAS methods utilize NAS-datasets that contains a large list of potential neural architectures from which we expect to find the most optimal architecture (for the target application) using the NAS method. One NAS dataset is the NAS-Bench-101[29] which contains 5 million distinct neural architectures and was designed for the CIFAR-10 dataset. The NAS-Bench-201[30] dataset has 15625 cell layouts and is derived from a cell-based search technique (for CIFAR-10[31], CIFAR-100[31] and ImageNet16-120[32] datasets). In[33], the authors proposed a NAS method named β-DARTS to solve the weak generalization ability found in the DARTS method. They used the NAS-Bench-201 to evaluate their framework. In another research work[34], the authors suggested Λ-DARTS as a solution for the structural flaws caused by the weight-sharing approach in DARTS. In a recent work[35], authors proposed GENIUS where they used an LLM to solve the NAS problem while utilizing a pre-defined search space and focusing solely on maximizing final model accuracy (no consideration given to model search efficiency or inferencing speed).

Shortcomings of NAS

Most traditional NAS techniques rely on having access to a pre-defined search space of potential neural architectures, making it difficult to scale across different applications and use cases. Additionally, most NAS frameworks do not have the capability to allow the search process to consider parameters such as: (1) Inferencing speed; (2) Inferencing energy consumption; (3) NAS search and training efficiency.

Why LLM and expert system for neural discovery?

We hypothesize that a large language model (LLM) trained on a large volume of open-domain data will also have the knowledge about different neural architectures. LLMs have demonstrated success in terms of searching for NN architectures given a search space[35–38]. However, we wanted to go one step further and find out if LLMs can generate novel NN architecture (discovery) without using a pre-defined search space. We also wanted to analyze if: (1) the open-domain knowledge has provided these LLMs with insights into different metrics associated with a neural architecture such as estimated training power consumption and inferencing speed; (2) these LLMs can follow automated instructions generated from an expert system for refining a NN. The abbreviations used in this study are listed in Table 1.

Abbreviation	Meaning	Location
LLM	Large language model	Absract
AI	Artificial intelligence	Absract
NN	Neural network	Introduction
FPS	Frames per second	Introduction
NAS	Neural architecture search	Introduction
LEMONADE	Large language model guided neural architecture discovery	Introduction
RL	Reinforcement learning	Neural architecture search
EA	Evolutionary algorithms	Neural architecture search
DNAS	Differentiable neural architecture search	Neural architecture search
GENIUS	GPT-4 enhanced neural architecture search	Neural architecture search
DARTS	Differentiable architecture search	Neural architecture search
ES	Expert system	Neural discovery process
TS	Task specification	Algorithm 1
TC	Termination condition	Algorithm 1
BCM	Best combine metric	Algorithm 1
cmd	Command	Algorithm 1
NNGES	Neural network generation expert system	Algorithm 1
AGI	Artificial generative intelligence	Algorithm 1
CM	Combine metric	Algorithm 1
NF	Normalized FPS	Equation 1
T_{NE}	Normalized training energy	Equation 1
V_{NE}	Normalized validation energy	Equation 1
Ins	Instructions	Algorithm 2

Table 1. Abbreviation.

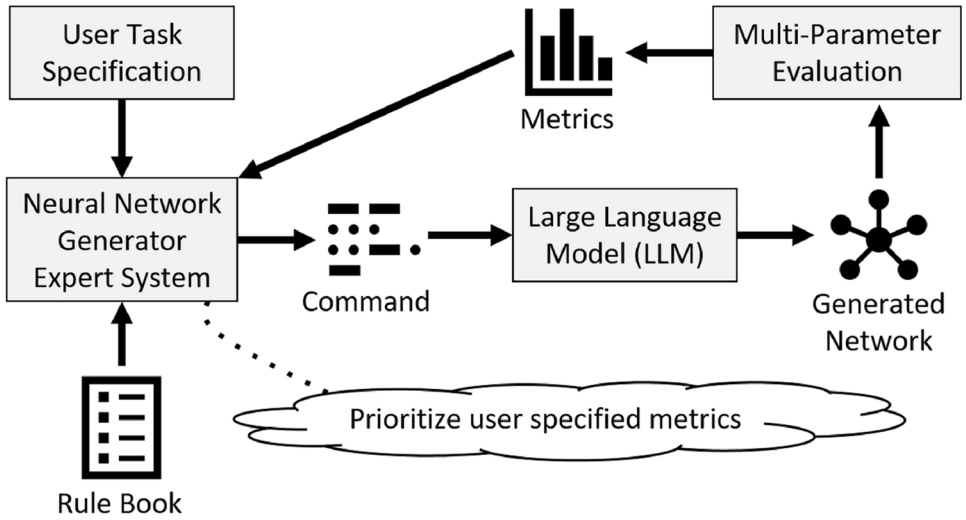

Fig. 1. LEMONADE framework: Expert system guided iterative and multi-parameter search for neural network discovery.

Methodology

Neural discovery process

In Fig. 1 we show an overview of the LEMONADE framework where an expert system (ES) takes the task specification from the user (metrics) and generates commands (prompt) for the LLM using a set of rules for creating a neural architecture. The generated neural network (in the form of python code) from the LLM is then trained on the training dataset and subsequently evaluated on the validation set. The associated evaluation-based metrics are used by ES to generate the next LLM prompt that modifies the current neural network architecture for improving overall efficacy.

Algorithm 1 shows the overall procedure of LEMONADE. *M* is a set of user-defined metrics, *TS* contains the task specification (classification/regression, input/output shape, datasets, etc), *TC* is an user defined terminating

condition (in our case a max number of iterations). From lines 2–5, all variables are initialized and the initial command is stored in the *cmd* variable. For example, the initial *cmd* might be something like - 'Please suggest a pytorch image classification model with input shape of (3,32,32) and output of 100'.

1: **function** AI_AG(M, TS, TC)
2: $BCM \leftarrow \emptyset$
3: $BestModel \leftarrow \emptyset$
4: $cmd \leftarrow generate\_Initial(TS, M)$
5: $TC = False$
6: **while** $TC = False$ **do**
7: Response ← LLM_AGI(cmd)
8: Model ← Create_Model(Response)
9: M.P ← Model.params
10: [Model, $M.T_E$, $M.T_{NE}$] ← Model.train(TS.train)
11: $M.T_{Acc}$ ← Model.eval(TS.train)
12: [$M.V_E, M.V_{NE}, M.V_{Acc}, M.NF$] ← Model.eval(TS.val)
13: cmd ← NNGES(M)
14: cmd ← RESOLVE_CONFLICT(cmd)
15: $CM \leftarrow Model\_Effectiveness(M)$
16: **if** $CM > BCM$ **then**
17: BCM ← CM
18: BestModel ← Model
19: **end if**
20: **if** Termination Condition is Met **then**
21: $TC = True$
22: **end if**
23: **end while**
24: **return** BestModel
25: **end function**

Algorithm 1. LEMONADE neural network discovery/search.

In line 7, LLM_AGI (GPT-4o[12] for this study) returns a response based on the command and subsequently, a model is created from the response.

From lines 10–12, the generated model is trained and evaluated with the training and validation datasets respectively and a set of metrics such as training energy expenditure (T_{NE}), training accuracy (T_{Acc}), validation energy expenditure (V_{NE}), validation accuracy (V_{Acc}), and validation set inferencing frames-per-second (*NF*) are calculated and stored in the Metrics dictionary. The *Metrics* dictionary is then passed to the NNGES (Fig. 2) to generate a set of instructions for the next round of LLM-based neural network generation.

Any conflicts between the generated instructions are removed using Algorithm 2. To identify the best network architecture, we utilize the following combined model effectiveness metric (*CM*):

$$CM = W_A \cdot (T_{Acc} + V_{Acc}) + (W_F \cdot NF) - W_E \cdot (T_{NE} + V_{NE}) \quad (1)$$

Where, W_A is weight for accuracy; T_{Acc} is training accuracy; V_{Acc} is validation accuracy; W_F is weight for FPS; *NF* is normalized FPS; W_E is weight for energy; T_{NE} is normalized training energy; V_{NE} is normalized validation energy. All parameters (user defined and evaluation-based) are described in Table 2.

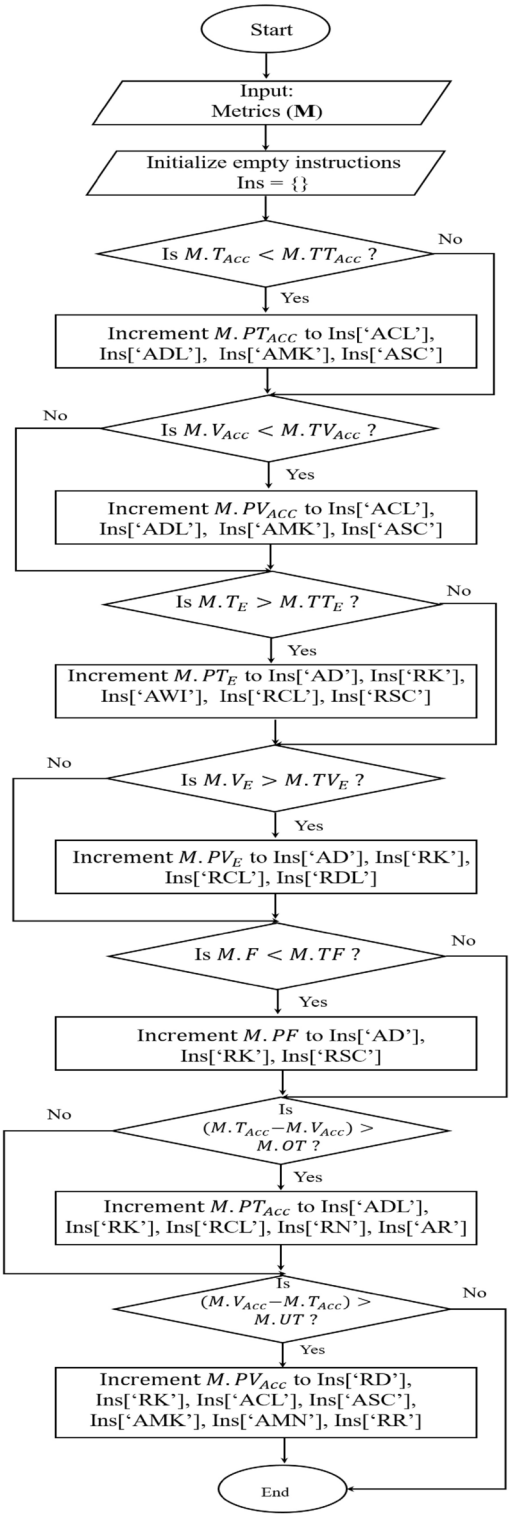

Fig. 2. Flowchart of the neural network generation expert system (NNGES).

Legends	Description	Type
T_{Acc}, V_{Acc}	Current model's training and validation accuracy respectively as predicted by LLM/AGI	Evaluation based
PT_{Acc}, PV_{Acc}	Priority of training and validation accuracy respectively	User defined
TT_{Acc}, TV_{Acc}	Threshold of the training and validation accuracy	User defined
T_E, V_E	Energy required for evaluating the training and validation set	Evaluation based
PT_E, PV_E	Priority of the energy required for evaluating the training and validation set	User defined
TT_E, TV_E	Threshold of energy required for evaluating the training and validation set	User defined
F	FPS of the current model predicted by LLM/AGI	Evaluation based
NF	Normalized FPS of the current model predicted by LLM/AGI	Evaluation based
PF	Priority of the FPS for the model	User defined
TF	Threshold ofthe FPS for the model	User defined
P	Parameters of the current model (CM) predicted by LLM/AGI	Evaluation based
OT, UT	Threshold value to check the overfitting and underfitting	User defined
W_A	Weight for the accuracy values when computing the combined metric (CM)	User defined
W_E	Weight for the energy values when computing the combined metric (CM)	User defined
W_F	Weight for the FPS values when computing the combined metric (CM)	User defined

Table 2. User-defined and evaluation based parameters.

```
function RESOLVE_CONFLICT(Ins)
    Refined_Ins ← ∅
    sort_descending(Ins)
    m ← 0
    key ← cmd.keys()
    while m < Length(Ins) do
        if Ins[key[m]] > 0 then
            Refined_Ins[key[m]] ← Ins[key[m]]
            n ← m + 1
            while n < Length(Ins) do
                if Ins[key[m]].conflict = Ins[key[n]] then
                    Ins[key[n]] ← 0
                end if
                n ← n + 1
            end while
        end if
        m ← m + 1
    end while
    return Refined_Ins
end function
```

Algorithm 2. LEMONADE conflict resolution.

Expert system for instruction set generation

We have developed an expert system for guiding the NAS process because: (1) The backend LLM can exhibit random behavior if proper bounds/rules are not set; (2) the backend LLM can start hallucinating during a lengthy neural search process and an expert system can keep it on track; (3) LLMs are not always aware of how to achieve a certain effect out of a neural network and requires additional input from an expert system to succeed.

The expert system utilizes a rulebook (Table 3) which is based on strategies that are commonly used by data scientists for constructing effective neural networks. In future, these rules can potentially be learned based on historical neural search data. The expert system drives the LLM towards constructing an optimal neural network for a given set of user-defined parameters by generating a series of instructions based on the Metrics calculated after each search iteration.

Figure 2 illustrates the flowchart of the Neural network generation expert system (NNGES) for instruction generation. The system takes M, a set of user-defined and evaluation metrics, as its input. It begins by initializing an empty dictionary for storing instructions. Next, the system compares the input metrics (M) with the user-defined metrics (see Table 2) and assigns different weights to each rule (as defined in Table 3) to construct an instruction dictionary. For instance, if the training accuracy ($M.T_{ACC}$) falls below the user-defined threshold ($M.TT_{ACC}$), the instruction assigns priority weights (PT_{ACC}) to Ins['ACL'], Ins['AMK'], Ins['ADL'], and

Legend	Description	Conflict with
ACL	Add convolutional layer	RCL
ASC	Add skip connection	RSC
ADL	Add dense layer	RDL
RCL	Reduce convolutional layer	ACL
RSC	Reduce skip connection	ASC
RDL	Reduce dense layer	ADL
AD	Add dropout layer	RD
AMK	Add more kernel	RK
AWI	Add weight initializer	–
AR	Add regularization	RR
RK	Reduce number of kernel	AMK
RD	Reduce dropout layer	AD
AMN	Add more neurons	RN
RN	Reduce neurons	AMN
RR	Reduce regularization	AR

Table 3. Rule book for the LEMONADE expert system.

Ins['ASC']. This implies that ACL, AMK, ADL, and ASC are expected to enhance training accuracy in the next iteration.

The system sequentially evaluates all conditions and stores the corresponding weights in the instruction dictionary, which will be further refined in the next section.

Conflict resolution

Instructions generation might have some conflicts (see Table 3) due to the nature of the instructions themselves. For example, instructions such as adding a dense layer (ADL) and removing a dense layer (RDL), may get assigned positive weights by the NNGES algorithm (e.g., 0.5 for ADL and 0.4 for RDL). However, since these instructions have opposite effects, both cannot be passed to the LLM simultaneously. To resolve this, Algorithm 2 is used to prioritize and select the instructions with higher weights. Algorithm 2 shows the overall procedure of eliminating conflicts. In line 3, we organize the *Ins* in descending order according to their values. Through lines 7–9, it identifies and stores the instructions that have values larger than 0 in $Refined\_Ins$. In lines 10–15, the algorithm examines the current instruction and the remaining instructions to identify any conflicts. If a conflict is detected, the algorithm looks at the computed metric to decide which one is more appropriate (by setting less appropriate metric to zero) for optimizing the NAS.

Dataset description and preparation

In this study, we have considered five publicly available image datasets: CIFAR-10[31], CIFAR-100[31], ImageNet16-120[32], EuroSAT[39], and Malaria Parasite[40] and a text dataset: IMDb[41] to validate our LEMONADE. Both the CIFAR-10 and the CIFAR-100 datasets contain 60k images of dimensions 32×32 pixels where 50k and 10k samples are designated for training and testing purposes, respectively. The CIFAR-10 dataset has 10 output classes whereas the CIFAR-100 datasets have 100 output classes. ImageNet16-120 has 151k training and 6k testing samples with a resolution of 16×16 distributed across 120 classes. The Malaria parasite dataset has two classes: (i) parasitized cells and (ii) uninfected cells with 27558 data samples in total. For our experiments with the Malaria parasite dataset, we resized the data to 32×32 resolution. After that we split it into an 8:2 ratio for the training and validation sets. The EuroSAT dataset contains 27000 data samples across 10 classes: (i) AnnualCrop (ii) Forest (iii) HerbaceousVegetation (iv) Highway (v) Industrial (vi) Pasture (vii) PermanentCrop (viii) Residential (ix) River, and (x) SeaLake. We also resized this dataset to 32×32 resolution and split it into 8:2 ratios for training and validation sets. Finally, the IMDb dataset contains 50k samples of text data with corresponding sentiment labels (positive and negative). We first split the data set into 8:2 ratio for training and validation and then pre-processed the text by removing urls, special characters, hashtags and mentions. Subsequently, we tokenize the text and transform it into sequences of fixed length.

Experimental analysis and results

All experiments are run on a single NVIDIA A100 GPU to make a fair determination of metrics such as power consumption and runtime. We utilize a python library named PyJoules[42] to measure the energy consumption of both CPU and GPU during network search/training. In this section, we will discuss the search strategy, the training process, and the experimental results. We use the following user defined metric for all experiments (unless something else is specifically mentioned): $\{PT_{Acc} = 0, PV_{Acc} = 1, TT_{Acc} = 0.99, TV_{Acc} = 0.99, PF = 0, TF = 14000, PT_E = 0, TT_E = 1 \times 10^{-3}, PV_E = 0, TV_E = 1 \times 10^{-5}, OT = 0.10, UT = 0.05\}$.

Intermediate and final model training process

During the neural discovery process, LEMONADE was executed for 30 iterations (Terminating Condition for Algorithm 1). To ascertain the quality of the searched network after each iteration, the searched networks are

trained for 50 epochs with a batch size of 128, utilizing the Stochastic Gradient Descent (SGD) optimizer along with an initial learning rate of 0.025 and a weight decay parameter set to 3×10^{-4}. To enhance the convergence rate, a cosine annealing learning rate schedule was employed, which modulates the learning rate according to a cosine function. Furthermore, to mitigate overfitting, data augmentation techniques such as random rotation by 10 degrees, random horizontal flipping, and random cropping of size 16×16 were integrated into the training regimen to facilitate diverse learning. Then the trained model is evaluated using the validation data (from the corresponding dataset) to obtain the combined metric (*CM*) as shown in Algorithm 1. For the NLP task (IMDb dataset), we used Adam optimizer along with a initial learning rate of 0.001 and batch size of 128.

Upon the completion of the NAS procedure by LEMONADE , the final model is trained on the entire dataset over 600 epochs (200 epochs for IMDb). A batch size of 256 is employed, utilizing the SGD optimizer (Adam for IMDB), with the initial learning rate set at 0.025 (0.001 for IMDb) accompanied by a weight_decay of 3×10^{-4}. The learning rate is systematically adjusted according to the annealing learning rate schedule. It is important to clarify that the complexity inherent in this final training phase is not inherently associated with the NAS process. Notably, larger datasets generally demand an increased number of training epochs. This is a challenge faced by all NAS frameworks and is not distinct to our methodology. In this experimental setup, GPT-4o was configured with a temperature parameter of 0.5.

Comparing LEMONADE with state-of-the-art NAS frameworks

Table 4 provides a comparative analysis of several State-of-the-Art (SOTA) NAS methods alongside LEMONADE , for the CIFAR-10, CIFAR-100, and ImageNet16-120 datasets. For these experiments we set $W_A = 1$, $W_E = 0$, and $W_F = 0$ because all the SOTA NAS we are comparing against only prioritize accuracy. For CIFAR-10, LEMONADE was able to generate a neural network with 95.54% test accuracy beating all SOTA frameworks. LEMONADE also beat state-of-the-art NAS frameworks for CIFAR-100 with a test accuracy of 79.43%. For ImageNet16-120, LEMONADE produced a neural network with almost SOTA performance. For the CIFAR-10 dataset, the NAS method (beside LEMONADE) that was able to achieve the highest accuracy was NSGANet[47]. But takes 648 GPU Hours for performing the search process compared to 5.8 GPU Hours that LEMONADE takes (111x Faster). For CIFAR-100, EIGEN[45] led the most efficient neural network (besides LEMONADE) but that search took 120 GPU Hour compared to 7.12 GPU Hours taken by LEMONADE. Hence, LEMONADE can not only discover highly accurate models, it can also perform search operations that are generally faster than many SOTA NAS frameworks. Figure 3 shows the training/validation accuracy and loss of five different datasets during the final model training process (over 600 epochs).

To better capture the efficacy of the NAS frameworks we report a joint metric that combines both cost (search time or energy consumed during search) and final model accuracy into one number that is weighted based on the user's need. This Goodness metric (*GM*) is computed as shown in Eq. 2. Where, X_A and X_E are the weights of accuracy and energy respectively. E_{Norm} is the normalized energy obtained from the Eq. 3. Where, E, E_{min}

Methods	Cost (GPU Hours)	Test accuracy (%)		
		CIFAR-10	CIFAR-100	ImageNet16-120
DeepMaker[43]	75	93.1	75.13	N/A
CGP-CNN[44]	744	94.05	73.3	N/A
EIGEN[45]	24	94.6	N/A	N/A
	120	N/A	78.1	N/A
GeNet[46]	408	92.9	70.95	N/A
NSGANet[47]	648	95.33	74.83	N/A
NASHBOT[48]	40.8	91.31	N/A	N/A
NASH-Net[49]	24	94.8	N/A	N/A
GDAS[50]	8.7	93.61	70.3	41.71
DARTS-[51]	3.2	93.8	71.53	45.12
GENIUS[35]	N/A	93.79 ± 0.09	70.91 ± 0.81	44.96 ± 1.02
DrNAS[52]	1.2	94.36	73.51	46.34
SE-NAS[53]	2.93	93.47 ± 0.14	N/A	45.66 ± 1.05
FairNAS[54]	2.73	93.23 ± 0.18	N/A	42.19 ± 0.31
Shapley-NAS[55]	7.2	94.37 ± 0.00	N/A	**46.85±0.12**
Distribution Constrained[56]	3.9	94.29 ± 0.07	N/A	46.41 ± 0.14
FreeRea[57]	N/A	94.36	N/A	46.34
RMI[58]	0.34	94.28 ± 0.10	73.36 ± 0.19	46.34 ± 0.00
LEMONADE (Proposed)	5.8	**95.54**	–	–
	7.12	–	**79.43**	–
	12.83	–	–	42.95

Table 4. Performance metrics for various methods across datasets. All results are for $W_A = 1$, $W_E = 0$, and $W_F = 0$. Significant values are in bold.

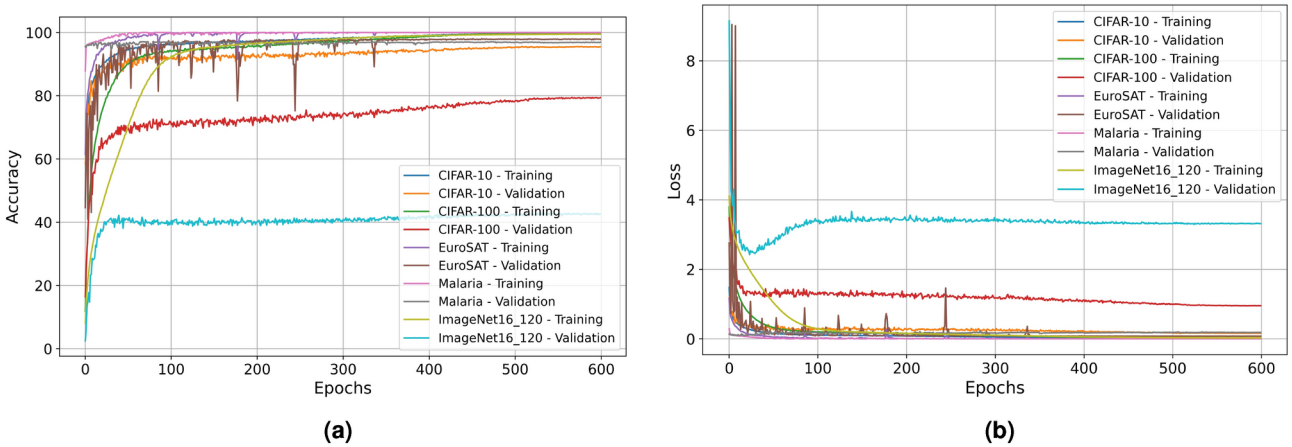

Fig. 3. Training process: loss and accuracy graph.

Method	Search energy (kWh-PUE)	*GM* values								
		$X_A = 0.8, X_E = 0.2$			$X_A = 0.5, X_E = 0.5$			$X_A = 0.2, X_E = 0.8$		
		CIFAR-10	CIFAR-100	ImageNet 16-120	CIFAR-10	CIFAR-100	ImageNet 16-120	CIFAR-10	CIFAR-100	ImageNet 16-120
DeepMaker[43]	35.55	0.925	0.781	–	0.915	0.825	–	0.906	0.870	–
CGP-CNN[44]	352.66	0.752	0.586	–	0.470	0.367	–	0.188	0.147	–
EIGEN[45]	9.48	0.952	–	–	0.960	–	–	0.968	–	–
	47.40	–	0.798	–	–	0.823	–	–	0.849	–
GeNet[46]	161.16	0.852	0.676	–	0.736	0.626	–	0.620	0.577	–
NSGANet[47]	307.15	0.788	0.624	–	0.541	0.439	–	0.294	0.253	–
NASHBOT[48]	19.34	0.920	–	–	0.929	–	–	0.939	–	–
NASH-Net[49]	11.38	0.952	–	–	0.958	–	–	0.964	–	–
GDAS[50]	4.12	0.947	0.760	0.531	0.962	0.846	0.703	0.978	0.932	0.874
DARTS-[51]	1.52	0.950	0.771	0.560	0.967	0.856	0.724	0.985	0.940	0.887
DrNAS[52]	0.57	0.955	0.788	0.570	0.971	0.867	0.731	0.988	0.946	0.892
SE-NAS[53]	1.39	0.947	–	0.565	0.966	–	0.727	0.984	–	0.889
FairNAS[54]	1.29	0.945	–	0.537	0.965	–	0.709	0.984	–	0.882
Shapley-NAS[55]	3.41	0.953	–	**0.573**	0.967	–	0.730	0.981	–	0.886
DC[56]	1.85	0.953	–	0.570	0.969	–	0.730	0.985	–	0.889
RMI[58]	0.16	0.954	0.787	0.571	0.971	0.867	**0.732**	**0.989**	0.947	**0.893**
LEMONADE (Proposed)	1.20	**0.964**	–	–	**0.976**	–	–	**0.989**	–	–
	2.34	–	**0.834**	–	–	**0.894**	–		**0.954**	–
	5.45	–	–	0.541	–	–	0.707	–	–	0.874

Table 5. Understanding the effectiveness of LEMONADE and other SOTA NAS frameworks utilizing the *GM* metric. Significant values are in bold.

and E_{max} are the energy consumed by a given NAS framework, maximum energy consumed for the same task across all NAS frameworks and minimum Energy consumed for the same task across all NAS frameworks (per Table 5). Kilowatt-hour power usage effectiveness (kWh-PUE) serves as a metric for evaluating the energy efficiency of an Edge AI system by comparing the overall energy consumed to that used specifically for AI inference/search/training[59]. We have calculated the energy (in kWh-PUE) with the help of Eq. 4 as described in[59]. In this equation, p_c, p_r, and p_g represent the power usages (in watt) of CPU, RAM and GPU respectively. Also, t is the total run time in hours and g is the number of GPUs. We obtain the run time for each NAS method from Table 4 and assume a maximum GPU power draw (in watt) for NVIDIA A100.

$$GM = X_A \times Accuracy + (1 - E_{Norm}) \times X_E \tag{2}$$

$$E_{Norm} = \frac{E - E_{min}}{E_{max} - E_{min}} \tag{3}$$

$$p_t = \frac{1.58t\,(p_c + p_r + gp_g)}{1000} \tag{4}$$

We observe a similar trend in Table 5 when comparing LEMONADE with other NAS frameworks. LEMONADE beats state-of-the-art NAS frameworks in terms of overall performance (*GM*) for CIFAR-10 and CIFAR-100 across different weight values. LEMONADE performs almost at the SOTA level for ImageNet16-120 as well.

Utilizing LEMONADE to construct neural networks for diverse datasets and application needs

It is evident that LEMONADE can very efficiently (cost and accuracy) generate neural networks with good accuracy for standard datasets with priority only given to final model accuracy. However, LEMONADE was designed to serve a more practical goal - *Automating the AI integration process for addressing diverse applications with varying needs*. LEMONADE is designed to help non-AI-experts build solutions for their respective tasks with varying needs such as high frame-per-second (FPS) and low energy energy inferencing for battery-operated edge devices. In Table 6, we demonstrate how LEMONADE can generate different neural networks for serving different application priorities (settings). For example, when we ask the LEMONADE system to give 100% priority ($W_A = 1$) to the final model accuracy (Malaria dataset), we obtain a neural network with an accuracy of 97.12% that consumes about 2.5×10^{-9} Kwh-PUE of energy for inferencing one image with an FPS of 157. For the same dataset (Malaria), if we make the LEMONADE system design a neural network with equal priority given to inferencing energy and accuracy ($W_A = 0.5, W_E = 0.5$) then we obtain a model that is 450x more energy efficient with about 1% lower accuracy.

To assess the generalizability of LEMONADE, we reported results across three different settings, similar to prior text classification, using the IMDb dataset in Table 6. When prioritizing accuracy at 100%, LEMONADE generates a model achieving 90.51% accuracy, 57 FPS, and an energy consumption of 8.42×10^{-9} kWh-PUE for per image inference. In contrast, with a priority distribution of 70% for accuracy, 10% for energy efficiency, and 20% for FPS, the generated model attains 89.03% accuracy, 473 FPS, and consumes only 6.19×10^{-12} kWh-PUE.

Table 7 shows the comparative analysis of different SOTA NAS with LEMONADE (using both chatGPT-4o and Gemini-Pro as backend). For comparison points, we consider different edge AI metrics i.e., test accuracy, required training time in hours, inference speed in milliseconds (ms), inference power in milliwatts (mW), and model size in MB. In the Table 7 setting represents the priority, where 1 refers to full priority on accuracy, 2 refers to 50% priority on accuracy and 50% priority on energy, and 3 refers to 70% priority on accuracy, 10% on energy, and 20% on FPS. For the malaria dataset, DARTS shows 96.61% accuracy with 26.1 ms inference speed and takes 799.58 mW power for inference. Where NasNet and AmoebaNet shows 96.88% and 97.09% accuracy with 25.07 ms and 22.29 ms inference speed, and 744.65 mW and 745.66 mW inference power respectively. LEMONADE with chatGPT-4o outperforming the three different NAS based on the accuracy, training time, and inference speed. The model size of LEMONADE is 37.53 MB which is larger than the SOTA NAS, this is because of the full priority given to accuracy. For setting-2 (50% priority to both accuracy and energy consumption), we notice that LEMONADE generates a light weight model with decent accuracy and faster inference speed. This is

Datasets	Settings	Frames per second	Inference energy per image (kWh-PUE)	Test accuracy (%)
CIFAR-10[31]	$W_A = 1$	814	2.83×10^{-10}	95.54
	$W_A = 0.5, W_E = 0.5$	524	7.23×10^{-10}	92.72
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	1606	3.10×10^{-11}	94.9
CIFAR-100[31]	$W_A = 1$	82	4.80×10^{-9}	79.43
	$W_A = 0.5, W_E = 0.5$	701	5.44×10^{-10}	68.6
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	950	3.56×10^{-10}	75.03
ImageNet16-120[32]	$W_A = 1$	1046	5.35×10^{-10}	42.95
	$W_A = 0.5, W_E = 0.5$	1147	5.11×10^{-11}	37.93
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	985	5.18×10^{-11}	38.2
Malaria[40]	$W_A = 1$	157	2.5×10^{-9}	97.12
	$W_A = 0.5, W_E = 0.5$	1517	5.56×10^{-12}	96.44
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	2506	7.70×10^{-12}	96.48
Euro-SAT[39]	$W_A = 1$	1468	1.23×10^{-10}	98.04
	$W_A = 0.5, W_E = 0.5$	1815	2.25×10^{-11}	95.7
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	1722	5.75×10^{-11}	97.48
IMDb[41]	$W_A = 1$	57	8.42×10^{-9}	90.51
	$W_A = 0.5, W_E = 0.5$	279	6.19×10^{-12}	89.03
	$W_A = 0.7, W_E = 0.1, W_F = 0.2$	473	2.03×10^{-9}	90.07

Table 6. Utilizing LEMONADE for building neural networks for diverse applications with different priorities and requirements..

Dataset	Method	Settings	Test accuracy (%)	Training time (hrs)	Training energy (kWh-PUE)	Inference speed (ms)	Inference power (mW)	Model size (MB)
Malaria[40]	DARTS[28]	–	96.61	6.11	2.14	26.1	799.58	13.64
	NasNet[60]	–	96.88	6.24	2.18	25.07	744.65	14.59
	AmoebaNet[24]	–	97.09	5.31	1.87	22.29	745.66	11.98
	LEMONADE(chatGPT-4o)	1	**97.12**	**2.57**	0.9	6.44	905.6	37.53
		2	96.48	2.82	**0.33**	0.66	19.21	**0.05**
		3	96.44	2.99	0.36	0.40	43.85	4.36
	LEMONADE(Gemini-Pro)	1	96.77	2.60	0.58	1.28	1192	43.66
		2	96.08	2.69	**0.33**	**0.37**	**6.69**	0.11
		3	96.50	2.62	0.44	0.56	760.08	6.18
EuroSAT[39]	DARTS[28]	–	96.76	5.28	1.83	23.88	780.48	12.93
	NasNet[60]	–	98.03	5.94	2.19	25.10	751.31	14.61
	AmoebaNet[24]	–	97.67	5.05	1.74	21.87	744.1	12.00
	LEMONADE(chat GPT-4o)	1	**98.04**	1.69	0.28	0.68	410.94	**0.95**
		2	95.70	1.62	**0.22**	0.55	93.02	15.95
		3	97.48	**1.50**	0.25	0.58	225.79	37.97
	LEMONADE(Gemini-Pro)	1	97.48	1.61	0.24	0.68	133.93	4.39
		2	95.56	1.72	0.23	**0.41**	**61.54**	3.42
		3	97.44	1.62	0.89	0.86	209.54	56.99

Table 7. In-depth analysis of LEMONADE with three different NAS with various priority settings: $\{1 \rightarrow (W_A = 1);\ 2 \rightarrow (W_A = 0.5, W_E = 0.5);\ 3 \rightarrow (W_A = 0.7, W_E = 0.1, W_F = 0.2)\}$ with consideration of edge AI metrics.. Significant values are in bold.

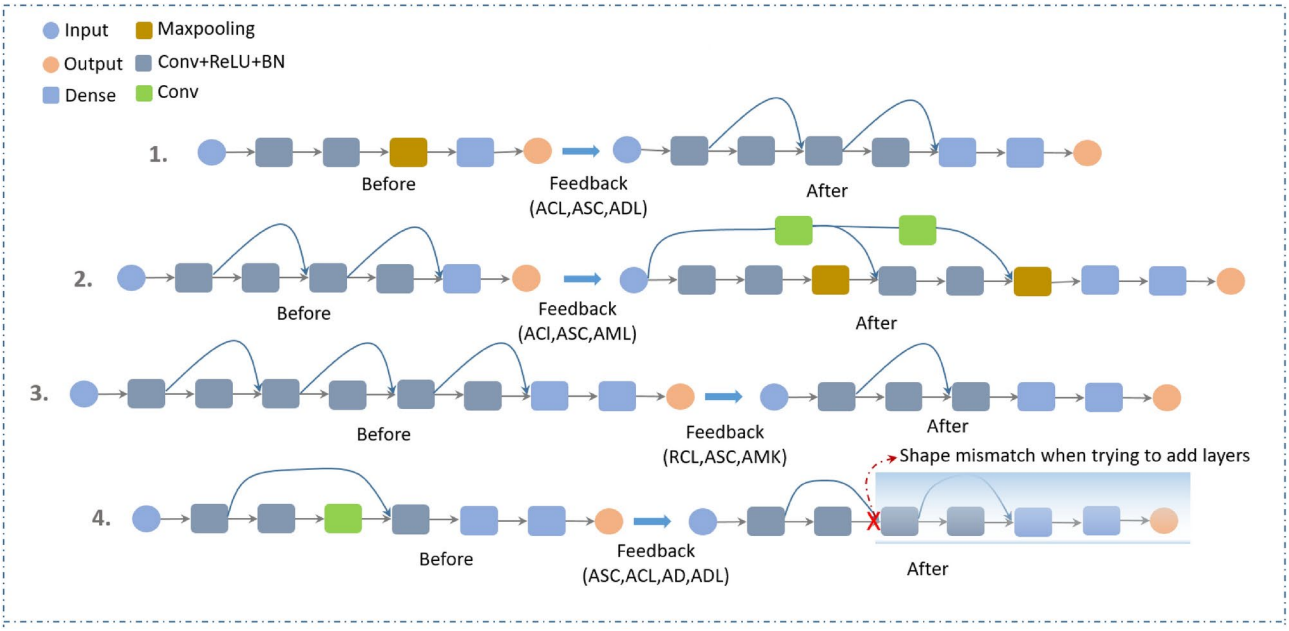

Fig. 4. Qualitative analysis of the effectiveness of LEMONADE with chatGPT-4o backend.

also noticeable for LEMONADE with the Gemini-Pro backend. For the EuroSAT dataset, the NasNet generated model shows almost equal accuracy with respect to LEMONADE (chatGPT-4o) but we see that LEMONADE is more efficient in terms of training time, inference time, and inference power.

In-depth analysis and limitations of LLM

The LEMONADE framework is also equipped with a post-processing module that ensures that the models received from the LLM is indeed valid. If an invalid model is received, LEMONADE invokes an LLM command to fix the identified issue. We illustrate such an example in Fig. 4.

We also showcase, with a few examples, how the the feedback mechanism of the LEMONADE's Expert System guides the LLM towards changing the generated neural network.

- *Case 1*: We can see that the initially generated model was a simple CNN model with two convolutional layers having ReLU activations and Batch Normalization (BN), Maxpooling, and a single Dense layer. After getting the feedback (ACL,ASC,ADL) chatGPT-4o generated a model with four blocks of convolutional layers with ReLU activation and Batch Normalization (BN), and two Dense layers. It also added two skip connections based on the feedback.
- *Case 2*: Shows another behavior of chatGPT-4o where it adds more skip connections by adding some convolutional layers.
- *Case 3*: In some situations, skip connections make the network architecture more energy intensive to train/run. In this case, chatGPT-4o reduces the skip connections based on the feedback from ES.
- *Case 4*: chatGPT-4o not always showed outstanding performance. We carefully checked the responses from chatGPT-4o and saw that in approximately 10% of the cases it fails to follow the provided instructions. In one case chatGPT-4o generated an invalid model that failed complication due to layer shape mismatch.

Limitations of using the GPT-4o/Gemini-Pro for NAS

ChatGPT-4o and Gemini-Pro can generate neural networks based on a prompt but they have several limitations that necessities the use of an additional automated guidance system (such as LEMONADE). We discuss some of these limitations below:

- *Prompt dependency* The effectiveness of a network architecture is significantly influenced by the quality of the prompt. A well-defined prompt enables GPT-4o or Gemini-Pro to produce network architectures that align with the specified requirements. Conversely, an ambiguous prompt may lead to outcomes that do not fulfill the objectives.
- *Inadequate validation capacity* Although the LLMs can propose a network structure, they lack the capability to independently train and validate the proposed architecture on a dataset.
- *Deficiency in numerical optimization* Current LLMs can not directly calculate the ideal hyperparameters of an architecture for the given task specifications.

Fine tuning GPT-4o/Gemini-Pro for the NAS

The following techniques have been used to improve the performance of the LLMs to generate good quality neural network architecture:

- *Prompt engineering:* Fine-tuning the prompt by specifying the constraints and validation outcomes helped us generate good-quality architecture.
- *Integrate expert system:* We have proposed an expert system that helps generate high quality architectures by providing structured feedback to the LLM backend (GPT-4o or Gemini-Pro).
- *Integrate external validation:* Since the LLMs lack the capability to train the proposed architectures, we have integrated a system for assessing various performance metrics such as accuracy, power, inference speed, and model size. These metrics were subsequently utilized in the next iteration's prompt, emulating a reinforcement approach.

Conclusion

In this article, we have formalized, implemented, and evaluated a multi-parameter neural discovery framework, LEMONADE that can efficiently generate novel neural networks for diverse requirements without leveraging any pre-defined search space. LEMONADE can effectively trade-off final model accuracy for other edge AI parameters such as FPS and inferencing energy cap. The proposed framework operates with the help of a set of customizable metrics and a rules-driven expert system. The proposed expert system generates instructions for a backend large language model (LLM) such as ChatGPT-4o and Gemini-Pro to iteratively produce novel neural networks. LEMONADE was able to successfully create state-of-the-art neural networks that are optimized for accuracy, FPS, and power consumption across different applications/requirements and datasets (CIFAR-10, CIFAR-100, ImageNet16-120, Malaria, Euro-SAT, IMDb). This work paves the way toward a new paradigm of AI-guided AI designing. Future works will investigate efficient model pruning and quantization using AI. Future works will also explore the use of a customized LLM that is specifically trained to generate AI models for a wider range of user-defined applications.

Data availability

The datasets analyzed in this study are publicly available in the following repository: The CIFAR-10 and CIFAR-100 datasets are available at https://www.cs.toronto.edu/~kriz/cifar.html. The ImageNet16-120 dataset is available at https://github.com/hafizuriu/ImageNet16. Malaria parasite data is available at https://www.kaggle.com/datasets/iarunava/cell-images-for-detecting-malaria Euro-SAT data is available at https://github.com/phelber/eurosat. IMDb data is available at https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews/data.

Received: 20 December 2024; Accepted: 3 April 2025
Published online: 15 May 2025

References

1. Chua, M. et al. Tackling prediction uncertainty in machine learning for healthcare. *Nat. Biomed. Eng.* **7**, 711–718 (2023).
2. Bhardwaj, R. & Tripathi, I. An enhanced reversible data hiding algorithm using deep neural network for e-healthcare. *J. Amb. Intell. Humaniz. Comput.* **14**, 10567–10585 (2023).
3. Nandy, S. et al. An intelligent heart disease prediction system based on swarm-artificial neural network. *Neural Comput. Appl.* **35**, 14723–14737 (2023).
4. Jaafar, N. & Lachiri, Z. Multimodal fusion methods with deep neural networks and meta-information for aggression detection in surveillance. *Expert Syst. Appl.* **211**, 118523 (2023).
5. Mahum, R. et al. A robust framework to generate surveillance video summaries using combination of zernike moments and r-transform and deep neural network. *Multimed. Tools Appl.* **82**, 13811–13835 (2023).
6. Jan, Z. et al. Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities. *Expert Syst. Appl.* **216**, 119456 (2023).
7. Raja Santhi, A. & Muthuswamy, P. Industry 5.0 or industry 4.0 s? Introduction to industry 4.0 and a peek into the prospective industry 5.0 technologies. *Int. J. Interact. Des. Manuf. (IJIDeM)* **17**, 947–979 (2023).
8. Shafiq, M. et al. Continuous quality control evaluation during manufacturing using supervised learning algorithm for industry 4.0. *Int. J. Adv. Manuf. Technol.* (2023).
9. Rajput, D. S., Meena, G., Acharya, M. & Mohbey, K. K. Fault prediction using fuzzy convolution neural network on IoT environment with heterogeneous sensing data fusion. *Meas. Sens.* **26**, 100701 (2023).
10. Liyakat, K. K. S. Machine learning approach using artificial neural networks to detect malicious nodes in IoT networks. In *International Conference on Machine Learning, IoT and Big Data* 123–134 (Springer, 2023).
11. Thakkar, A. & Lohiya, R. Attack classification of imbalanced intrusion data for IoT network using ensemble learning-based deep neural network. *IEEE Internet Things J.* **10**, 11888–11895 (2023).
12. OpenAI, R. Gpt-4 technical report. Preprint at arxiv:2303.08774. *View in Article***2**, 13 (2023).
13. Wang, J. et al. El-nas: Efficient lightweight attention cross-domain architecture search for hyperspectral image classification. *Remote Sens.* **15**, 4688 (2023).
14. Yang, T., He, Q. & Huang, L. OM-NAS: Pigmented skin lesion image classification based on a neural architecture search. *Biomed. Opt. Express* **14**, 2153–2165 (2023).
15. Yang, Y., Wei, J., Yu, Z. & Zhang, R. A trustworthy neural architecture search framework for pneumonia image classification utilizing blockchain technology. *J. Supercomput.* **80**, 1694-1727 (2024).
16. Hassan, E. et al. Mask r-CNN models. *Nile J. Commun. Comput. Sci.* **3**, 17–27 (2022).
17. Dong, P. et al. Rd-nas: Enhancing one-shot supernet ranking ability via ranking distillation from zero-cost proxies. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 1–5 (IEEE, 2023).
18. Wang, J. et al. Nas-dymc: Nas-based dynamic multi-scale convolutional neural network for sound event detection. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 1–5 (IEEE, 2023).
19. Li, J. et al. Graph neural network architecture search for rotating machinery fault diagnosis based on reinforcement learning. *Mech. Syst. Signal Process.* **202**, 110701 (2023).
20. Yuan, W., Fu, C., Liu, R. & Fan, X. SSoB: Searching a scene-oriented architecture for underwater object detection. *Vis. Comput.* **39**, 5199–5208 (2023).
21. Jia, X. et al. Fast and accurate object detector for autonomous driving based on improved yolov5. *Sci. Rep.* **13**, 1–13 (2023).
22. Mehta, R., Jurečková, O. & Stamp, M. A natural language processing approach to malware classification. *J. Comput. Virol. Hacking Tech.* **20**, 173-184 (2024).
23. Girdhar, N., Coustaty, M. & Doucet, A. Benchmarking nas for article separation in historical newspapers. In *International Conference on Asian Digital Libraries*, 76–88 (Springer, 2023).
24. Real, E., Aggarwal, A., Huang, Y. & Le, Q. V. Regularized evolution for image classifier architecture search. *In Proceedings of the AAAI conference on artificial intelligence* vol. 33, 4780–4789 (2019).
25. Liu, C. et al. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)* 19–34 (2018).
26. Cai, H., Chen, T., Zhang, W., Yu, Y. & Wang, J. Efficient architecture search by network transformation. In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32 (2018).
27. Pham, H., Guan, M., Zoph, B., Le, Q. & Dean, J. Efficient neural architecture search via parameters sharing. In *International conference on machine learning* 4095–4104 (PMLR, 2018).
28. Liu, H., Simonyan, K. & Yang, Y. Darts: Differentiable architecture search. Preprint at arXiv:1806.09055 (2018).
29. Ying, C. et al. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning* 7105–7114 (PMLR, 2019).
30. Dong, X. & Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. Preprint at arXiv:2001.00326 (2020).
31. Krizhevsky, A. & Hinton, G. *Learning multiple layers of features from tiny images* (Tech. Rep, Toronto, ON, Canada, 2009).
32. Chrabaszcz, P., Loshchilov, I. & Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. Preprint at arXiv:1707.08819 (2017).
33. Ye, P. et al. β-darts: Beta-decay regularization for differentiable architecture search. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* 10864–10873 (IEEE, 2022).
34. Movahedi, S. et al. λ -darts: Mitigating performance collapse by harmonizing operation selection among cells. Preprint at arXiv:2210.07998 (2022).
35. Zheng, M. et al. Can GPT-4 perform neural architecture search? Preprint at arXiv:2304.10970 (2023).
36. Achiam, J. et al. GPT-4 technical report. Preprint at arXiv:2303.08774 (2023).
37. Wang, H. et al. Graph neural architecture search with GPT-4. Preprint at arXiv:2310.01436 (2023).
38. Hassan, E., Bhatnagar, R. & Shams, M. Y. Advancing scientific research in computer science by ChatGPT and llama-a review. In *International conference on intelligent manufacturing and energy sustainability* 23–37 (Springer, 2023).
39. Helber, P., Bischke, B., Dengel, A. & Borth, D. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **12**(7), 2217–2226 (2019).
40. Rajaraman, S. et al. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ* **6**, e4568 (2018).
41. Maas, A. et al. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* 142–150 (2011).
42. PowerAPI. Pyjoules: Python-based energy measurement library for various domains including nvidia gpus. https://github.com/powerapi-ng/pyJoules (2024). Accessed: 2024-05-31.
43. Loni, M., Sinaei, S., Zoljodi, A., Daneshtalab, M. & Sjödin, M. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsyst.* **73**, 102989 (2020).
44. Suganuma, M., Kobayashi, M., Shirakawa, S. & Nagao, T. Evolution of deep convolutional neural networks using cartesian genetic programming. *Evol. Comput.* **28**, 141–163 (2020).
45. Ren, J. et al. Eigen: Ecologically-inspired genetic approach for neural network structure searching from scratch. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* 9059–9068 (2019).

46. Xie, L. & Yuille, A. Genetic CNN. In *Proceedings of the IEEE international conference on computer vision* 1379–1388 (2017).
47. Lu, Z. et al. Multi-criterion evolutionary design of deep convolutional neural networks. Preprint at arXiv:1912.01369 (2019).
48. Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B. & Xing, E. P. Neural architecture search with bayesian optimisation and optimal transport. *Adv. Neural Inf. Process. Syst.* **31** (2018).
49. Elsken, T., Metzen, J.-H. & Hutter, F. Simple and efficient architecture search for convolutional neural networks. Preprint at arXiv:1711.04528 (2017).
50. Dong, X. & Yang, Y. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition* 1761–1770 (2019).
51. Chu, X. et al. Darts-: Robustly stepping out of performance collapse without indicators. Preprint at arXiv:2009.01027 (2020).
52. Chen, X., Wang, R., Cheng, M., Tang, X. & Hsieh, C.-J. Drnas: Dirichlet neural architecture search. Preprint at arXiv:2006.10355 (2020).
53. Hu, Y., Wang, X., Li, L. & Gu, Q. Improving one-shot NAS with shrinking-and-expanding supernet. *Pattern Recogn.* **118**, 108025 (2021).
54. Chu, X., Zhang, B. & Xu, R. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF international conference on computer vision* 12239–12248 (2021).
55. Xiao, H., Wang, Z., Zhu, Z., Zhou, J. & Lu, J. Shapley-NAS: Discovering operation contribution for neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* 11892–11901 (2022).
56. Yu, K., Ranftl, R. & Salzmann, M. Landmark regularization: Ranking guided super-net training in neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* 13723–13732 (2021).
57. Cavagnero, N., Robbiano, L., Caputo, B. & Averta, G. Freerea: Training-free evolution-based architecture search. In *Proceedings of the IEEE/CVF Winter conference on applications of computer vision* 1493–1502 (2023).
58. Zheng, X. et al. Neural architecture search with representation mutual information. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* 11912–11921 (2022).
59. Strubell, E., Ganesh, A. & McCallum, A. Energy and policy considerations for deep learning in NLP. Preprint at arXiv:1906.02243 (2019).
60. Zoph, B., Vasudevan, V., Shlens, J. & Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 8697–8710 (2018).

Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. DRL-2342746 and Grant No. OIA-2416915.

Author contributions

M.H.R. was responsible for designing the experiment, analyzing the data, preparing the results, and writing the manuscript. Z.H. conducted the experimental analysis and contributed to writing the manuscript. P.C. conceptualized the idea, contributed to the experimental design, and also participated in writing the manuscript.

Declarations

Competing interests

The authors declare that they have no competing interests.

Additional information

Correspondence and requests for materials should be addressed to M.H.R.