LEMUR 2: Unlocking Neural Network Diversity for AI

Tolgay Atinc Uzun*, Waleed Khalid, Saif U Din, Sai Revanth Mulukuledu, Akashdeep Singh, Chandini Vysyaraju, Raghuvir Duvvuri, Avi Goyal, Yashkumar R. Lukhi, Muhammad A. Hussain, Krunal Jesani, Usha Shrestha, Yash Mittal, Roman Kochnev, Pritam Kadam, Mohsin Ikram, Harsh R. Moradiya, Alice Arslanian, Dmitry Ignatov†, Radu Timofte

Computer Vision Lab, CAIDAS & IFI, University of Würzburg, Germany

Corresponding authors: *tolgay-atinc.uzun@stud-mail.uni-wuerzburg.de, †dmytro.ignatov@uni-wuerzburg.de

https://github.com/ABrain-One/NN-Dataset

Abstract

Existing NAS benchmarks (e.g., NAS-Bench, NATS-Bench) cover only narrow, task-specific regions of the architectural design space and lack cross-domain or deployment-aware evaluation. LEMUR 2 introduces a large-scale, extensible framework unifying generative, evaluative, and deployment pipelines to unlock neural-network diversity. It comprises over 14,000 distinct architectures and more than 750,000 structured training records documenting model performance, hyperparameters, and task outcomes. These models were produced through AST-based code mutation, genetic and reinforcement-learning evolution, generation of fractal architectures, and synthesis guided by a Large Language Model (LLM). This includes deep models generated with the retrieval-augmented system NN-RAG, which derived and used architectural motifs from over 900 PyTorch modules extracted from public repositories. LEMUR 2 further employs NN-VR and NN-Lite pipelines for automated deployment and latency benchmarking on heterogeneous mobile and Unity-based VR platforms, providing real-device performance metadata. It spans multimodal tasks—image captioning, text-to-image synthesis, and language modeling—supporting cross-domain analysis of architectural transferability. By linking diverse architectures, tasks, and deployment data, LEMUR 2 provides the data foundation for LLM fine-tuning, coupling diverse architectural origins with large-scale, cross-platform empirical validation. This dataset supports reproducible, data-driven AI design, advancing LLM-driven AutoML and cross-modal, hardware-aware architectural generalization.

1. Introduction

Neural networks underpin numerous breakthroughs in artificial intelligence, delivering state-of-the-art results in fields such as computer vision and natural language processing. Their growing complexity, however, has exposed the limits of manual design. In response, the field is shifting toward a paradigm centered on creating and leveraging large-scale, reusable collections of neural network models. The goal of this shift is to replace manual intuition with data-driven discovery, enabling researchers to identify latent patterns, structural regularities, and generalizable design principles from diverse model corpora. Such model-centric datasets are now critical resources for benchmarking, neural architecture analysis, meta-learning, and automated machine learning.

Most publicly available neural architecture repositories are built around a single construction paradigm, typically exhaustive or near-exhaustive enumeration of a fixed cell-based search space. This yields methodologically homogeneous collections that cover only a narrow region of the architectural design space and are not readily extensible to models obtained by other means. At the same time, these repositories rarely provide an end-to-end path from architectural specification to task-level evaluation and device-level deployment, making it difficult to study accuracy-latency trade-offs or reuse models across modalities.

We propose multiple strategies that build on the LEMUR [15] dataset and accompanying pipeline to improve architectural diversity and operational integration. To generate more models, we employ programmatic editing, reinforcement learning, evolution-based search, fractal construction, retrieval-augmented block extraction, and LLM-driven synthesis. On the deployment side, we provide NN-Lite and NN-VR for benchmarking, testing, and verification across multiple tasks on resource-constrained targets under a shared protocol. Statistics are available in the repository of the NN-Dataset project.

1.1. Related Work

In neural network research, datasets are the backbone of training, evaluation, and benchmarking. There are plentiful datasets in image, text, and domain-specific corpora [31] that provide large, labeled collections of data. By contrast, resources that systematically capture the structures, configurations, and resulting performance of neural networks themselves are far less common.

Dataset of models The closest line of work comes from the neural architecture search (NAS) community. Benchmarks such as NAS-Bench-101 [50] and its extension NAS-Bench-201 [7] enumerate a fixed, small search space of convolutional cells and store their training and evaluation results to enable reproducible comparison of NAS algorithms. NATS-Bench [8] generalizes this idea to both topology and size spaces across multiple datasets, while TransNAS-Bench-101 [9] expands it to several vision-style tasks to study transferability. NAS-Bench-NLP [24] moves beyond vision to language models, HW-NAS-Bench [29] adds hardware and latency measurements, and JAHS-Bench-201 [2] couples architectures with hyperparameters for joint optimization.

AutoML Frameworks and Model Repositories AutoML frameworks such as AutoKeras [22] and TPOT [34] provide tooling for automated model selection and hyperparameter optimization but do not maintain large-scale, curated repositories of architectures with performance metadata. Similarly, model zoos like TensorFlow Hub [16] and PyTorch Hub [37] offer pre-trained models but lack standardized evaluation across tasks and hardware, and they do not support systematic architecture generation or diversity analysis.

1.2. Our Contribution

This paper presents LEMUR 2, an extension of the original LEMUR [15] framework. While LEMUR served as a repository for existing neural networks and their associated statistics, inspired by recent advancements in the use of LLMs across various domains [14, 25, 43], LEMUR 2 is designed to include generative systems that leverage existing data for the automated creation of new network architectures [11] as well as an assessment tool for testing on edge devices [6] and deployment on different platforms.

First, the dataset was expanded to over 14,000 models using several distinct, automated generation methodologies. These methods include genetic algorithms, reinforcement learning-based layer masking for network mutation, abstract syntax tree (AST) editing of model source code, and fractal-inspired generation. To facilitate dynamic collection, we also introduce NN-RAG, a retrieval-augmented system that extracts validated, self-contained PyTorch [36] modules from external codebases, creating a library of over 900 reusable components. The dataset is further enriched with metadata from a systematic evaluation of 6,000 data transformation pipelines, providing guidance on optimal data preprocessing. The framework's applicability is demonstrated through the inclusion of additional tasks, such as image captioning, text-to-text and text-to-image synthesis.

Second, to connect theoretical metrics with practical performance, we added deployment-aware metadata. We built NN-Lite, an automated pipeline that converts, deploys, and benchmarks PyTorch models on the Android platform. This system processed over 7,500 models, populating the LEMUR 2 database with their on-device inference latencies, empirical measurements often absent from standard benchmarks. This deployment analysis was extended to immersive applications with NN-VR, a system for the automated conversion and performance evaluation of models within the Unity engine for virtual reality contexts.

Collectively, these contributions establish LEMUR 2 as a public research resource. It provides a large-scale architectural corpus, a set of generative methodologies, and multi-platform deployment benchmarks that support research in automated AI design. The empirical value of the LEMUR dataset, as demonstrated within the NNGPT project, is proved by its successful application to LLM-guided generative synthesis, exploration of complex architectural structures, and improved generalization across multiple computer vision tasks [13, 17, 21, 25, 33, 45, 47, 48].

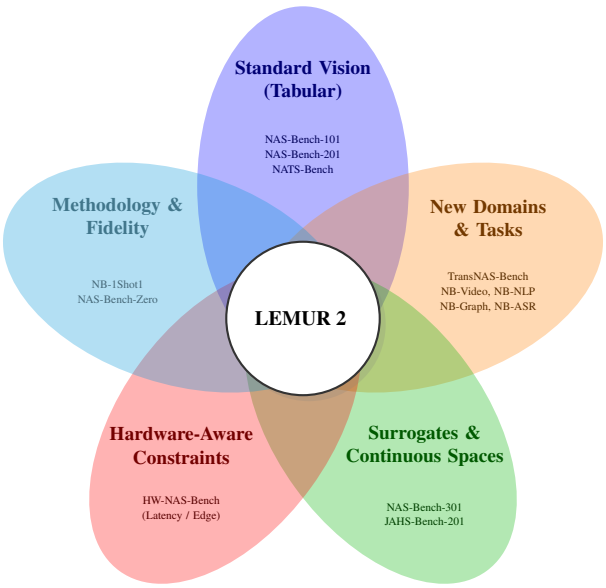

Figure 1. Taxonomy of NAS benchmarks organized by their primary differentiating dimension. **LEMUR 2** lies at the intersection, providing a unified generative framework that spans tabular evaluation, methodological abstractions, surrogate modeling, hardware awareness, and multi-domain tasks.

Table 1. Comparison of prominent NAS benchmarks. #Entries denotes the number of distinct queryable model specifications released by the benchmark (architectures, or architecture+HP configurations), each with stored trained or predicted metrics. Multi-Task indicates whether multiple datasets and/or multiple tasks are included within the benchmark's domain.

							Capabilities
Benchmark	**Domain**	**Search Space**	**#Entries**	**Datasets**	**Tasks**	**Evaluation**	**Multi-Task**
NAS-Bench-101 [50]	Vision	Cell tabular	423,624	CIFAR-10	Image classification	Trained	×
NAS-Bench-201 [7]	Vision	Cell tabular	15,625	CIFAR-10, CIFAR-100, ImageNet16-120	Image classification	Trained	✓
NAS-Bench-301 [46]	Vision	Cell surrogate (DARTS)	∼60,000	CIFAR-10	Image classification	Predictive	×
NATS-Bench [8]	Vision	Topology+size tabular	48,393	CIFAR-10, CIFAR-100, ImageNet16-120	Image classification	Trained	✓
TransNAS-Bench [9]	Vision	Cell+macro tabular	7,352	Taskonomy (subset)	7 vision tasks	Trained	✓
NAS-Bench-NLP [24]	NLP	RNN tabular	14,322	PTB; WikiText-2 (subset)	Language modeling	Trained	✓
NAS-Bench-Graph [38]	Graph	GNN tabular	26,206	Cora, CiteSeer, PubMed Coauthor-CS/Physics, Amazon-Photo/Computers ogbn-arXiv, ogbn-proteins	Node classification	Trained	✓
NAS-Bench-ASR [32]	Audio	ASR cell tabular	8,242	TIMIT	Speech recognition	Trained	×
HW-NAS-Bench [29]	Hardware	NB201 + FBNet (HW)	15,625	CIFAR-10, CIFAR-100, ImageNet16-120	HW cost (latency+energy; 6 devices)	Trained + Measured	✓
NAS-Bench-1Shot1 [51]	Method	One-shot tabular subsets	399,048	CIFAR-10	Image classification	Trained	×
JAHS-Bench-201 [2]	HPO	Arch+HP surrogate	270,000	CIFAR-10, Fashion-MNIST, Colorectal Histology	Image classification + HPO	Predictive	✓
LEMUR 2 (Ours)	**Universal**	**Open code (LLM)**	**>14,000**	**Multiple (user-defined)**	**Multi-domain**	**Trained**	✓

2. Methodology

We extend the LEMUR [15] corpus as a modular generation and integration framework built on the original experiment manager for task execution and result persistence. Corpus access and downstream processing use the `data` interface, which exposes stored records as a fixed-schema pandas `DataFrame` for uniform filtering and aggregation across tasks, datasets, metrics, architectures, epochs, accuracy, runtime, parameter counts, and transformation identifiers without direct database queries. On this layer, self-contained extension modules generate candidate model definitions or transformation programs with metadata and submit them to the standard LEMUR pipeline for evaluation. These modules combine LLM-based generation from natural-language guidance with programmatic generators that produce valid variants faster and at lower cost, enabling scalable, heterogeneous expansion under a shared reporting format.

Architectural diversity in LEMUR 2 comes from combining heterogeneous generators and sources, so the corpus is not centered on a single backbone family. For generators that operate by mutating a seed architecture, we instantiate them on a lightweight reference template supported by the generator to enable high-throughput evaluation and controlled attribution of operator effects, while still producing many structurally distinct variants.

2.1. Task Extensions

2.1.1. Image Captioning

NN-Caption is an automated image-captioning pipeline built on LEMUR in which an LLM, iteratively prompted with a baseline ResNet–LSTM captioner and code templates for the LEMUR `Net` interface (`__init__`, `train_setup`, `learn`, `forward`), generates PyTorch encoder–decoder architectures that couple a convolutional image encoder with sequence decoders (LSTM, GRU [5], Transformer). Generated code is normalized, validated via Python's abstract syntax tree, integrated into LEMUR, compiled, and trained on MS COCO [31], with caption quality evaluated using BLEU-4 [35]; in total, NN-Caption contributes 357 captioning architectures to LEMUR 2.

2.1.2. Text-to-Image

LEMUR includes a text-to-image pipeline spanning three generative families: a diffusion model (UNet-D) [18, 41, 42], a GAN [52], and a CVAE-GAN [23], all conditioned on natural-language text within a unified training and evaluation framework. For diffusion, a pre-trained CLIP text encoder [39] provides embeddings that are fused with time-step embeddings and injected into a multi-scale U-Net encoder–decoder with residual and attention blocks, optimized with a denoising diffusion objective. The GAN uses a generator that fuses text representations with a convolutional upsampling stack and a discriminator operating on joint image–text pairs, whereas the CVAE-GAN employs a conditional VAE whose decoder serves as the generator alongside an adversarial discriminator that promotes sharp outputs. All models are trained on paired text–image data with architecture-specific losses (diffusion, adversarial, or variational plus adversarial), and performance is quantified via CLIP-based text–image similarity between generated images and their prompts.

2.1.3. Text-to-Text

LEMUR is extended to natural language generation via a text-to-text pipeline for training and evaluating recurrent language models with standardized metrics. A lightweight WikiText data loader tokenizes text into integer sequences and forms batched streams for truncated backpropagation through time [49], with context length, vocabulary size, and batching strategy configured via a shared dictionary. Two recurrent baselines are instantiated: a stacked Elman RNN [10] with learned token embeddings, `tanh` activations, and a linear next-token decoder, and a stacked LSTM [19] with input, forget, and output gates that improve

stability and long-range dependency modeling. Evaluation uses perplexity and BLEU, with all LEMUR metrics normalized to $[0, 1]$; BLEU is computed at corpus level with smoothed n-gram precision on the validation set.

2.1.4. Mixture-of-Experts (MoE)

Mixture-of-Experts architectures are integrated into LEMUR to study expert composition and routing strategies on CIFAR-10 [26]. Eight variants are constructed in three stages: (i) homogeneous MoEs based on a single backbone type, (ii) AlexNet-based MoEs, and (iii) a heterogeneous MoE combining multiple backbones.

Homogeneous configurations use identical expert architectures and a sparse routing mechanism. For each input, a gating network selects the two experts with the highest scores (top-2 routing), and only these experts are evaluated. Heterogeneous configurations combine four different backbone architectures—AlexNet [27], AirNet [4], DenseNet [20], and BagNet [3]—and employ a soft routing strategy in which the gating network produces a weight for each expert. The final prediction is obtained as a weighted aggregation of the experts' outputs. All MoE variants are trained and evaluated within the LEMUR framework using the standard CIFAR-10 training protocol.

2.2. Achieving Diversity

2.2.1. AST-Based Mutations

An AST-based mutation mechanism[47] generates structurally consistent architectural variants by modifying the source code of existing models while operating on the channel dimensions of convolutional and linear layers and preserving tensor-shape compatibility. A symbolic tracer built on `torch.fx` [40] constructs a computation graph whose nodes represent layers with explicit dataflow edges, and maintains a source map from each module instantiation to its file location. Given a target layer, a planning component uses this graph to identify all dependent consumer layers and infer required input/output channel adjustments, including in architectures with residual connections. The mutation plan is applied by parsing the implementation into an AST, locating the corresponding constructor calls via the recorded coordinates, and updating channel-related arguments; the modified AST is then rendered back to source code, re-imported, and each mutant is validated via instantiation and forward/backward passes on dummy inputs. This procedure yields approximately 1000 additional architectures in the corpus.

2.2.2. Reinforcement Learning

The LEMUR dataset is processed by programmatically parsing each network's source code and identifying layer-instantiation statements (e.g., `nn.Conv2d`, `nn.Linear`) within the `__init__` method. These code blocks are removed and replaced with placeholder tokens, while preserving the surrounding class structure, method signatures, and forward pass logic. This produces a dataset of masked model skeletons paired with their complete source code as ground truth for training.

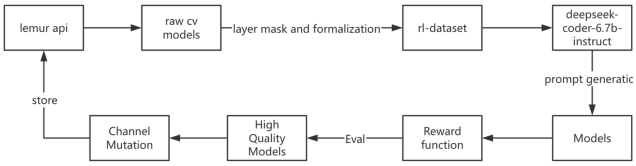

Figure 2. Reinforcement Learning pipeline overview. Using the data from LEMUR, the networks are masked to construct a training dataset. By deriving the policy from LLM, generated sequences of models are assessed and rewarded.

The reinforcement learning pipeline then iteratively refines architecture generation by completing these masked skeletons. The LLM acts as a policy, generating code completions (actions) given the masked skeleton (state). Each generated model undergoes validation checks for compilation and tensor shape consistency via a dummy forward pass. Successful models are trained for one mini-epoch on CIFAR-10 to measure accuracy.

A composite reward is calculated: -1.0 for validation failure, $+0.2$ for each passed stage (compilation, forward pass, training), and $+1.0 \times \Delta accuracy$ for improvement over baseline. This reward updates the LLM's policy using Group Relative Policy Optimization (GRPO) [44].

2.2.3. Genetic Algorithm

A genetic algorithm explores AlexNet-style architectures on CIFAR-10 [26] using a parameterized representation that includes architectural and training hyperparameters. Populations of candidate networks are evolved over generations via crossover and mutation on these parameter vectors; fitness is approximated by training each model for a small number of epochs and using validation accuracy for selection, with a checksum over the architecture description used to discard duplicates and maintain diversity. Two search spaces are considered: one restricted to hyperparameters (e.g., filter counts, kernel sizes, learning rate, dropout), and one that additionally includes block-level structural choices (pooling type, activation function, batch normalization). Across both spaces, the algorithm yields approximately 2,000 distinct AlexNet-type architectures.

2.2.4. Fractal-Inspired Computational Architectures

LEMUR includes a FractalNet-style [28] generator [33] that constructs self-similar, recursively defined multi-column networks whose topology is governed by fractal depth N (recursion level) and column width (number of parallel pathways), enabling balanced growth in depth and width. The pipeline comprises (i) configuration generation, which systematically samples architectural blueprints via permutations of convolution, normalization, activation,

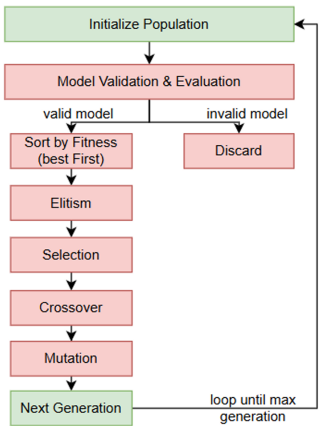

Figure 3. Genetic algorithm model pipeline overview.

and dropout layers; (ii) template-based model instantiation, which programmatically realizes the specified recursive, multi-column PyTorch architectures; and (iii) automated evaluation, which executes a standardized training and logging protocol. To scale to over 1,200 unique architectures, training employs Automatic Mixed Precision (AMP) to accelerate computation with half-precision arithmetic and gradient checkpointing to reduce GPU memory via recomputation of intermediate activations.

2.2.5. Retrieval-Augmented Generation

The NN-RAG component [12, 13] constructs a library of reusable PyTorch modules by mining existing codebases. It scans repositories for subclasses of `torch.nn.Module` that implement a `forward()` method and extracts them as self-contained units that preserve the original imports and semantics.

Source files are parsed using LibCST [30], which maintains syntactic structure, formatting, and comments. A scope-sensitive dependency resolver computes, for each candidate module, the minimal transitive closure of required definitions (such as auxiliary classes, functions, and constants) without executing the code. The resulting set of definitions is topologically ordered to ensure definition-before-use and is written out as an isolated module. Each extracted module passes through a multi-stage validation pipeline comprising abstract syntax tree parsing, bytecode compilation, and sandboxed execution to detect syntax and import-time errors. Valid modules are then integrated into LEMUR as reusable building blocks, with associated metadata linking them to their source repositories.

2.2.6. Few-Shot Architecture Prompting

To improve the stability of LLM-based architecture generation, we employ few-shot prompting and deduplication. Prompts include a small set of high-performing LEMUR architectures as exemplars, together with a description of the target dataset, and the LLM is instructed to produce

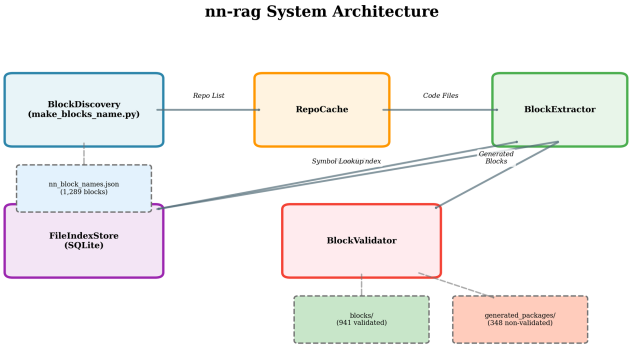

Figure 4. NN-RAG pipeline overview.

a new architecture consistent with the LEMUR interface; the number of exemplars is treated as a tunable parameter. Each generated architecture is normalized into a canonical form and assigned a whitespace-stripped MD5 hash, which serves as a database key for fast detection and removal of exact duplicates prior to training. Non-duplicate architectures are trained under the standard LEMUR protocol and their performance recorded. For statistical analysis, evaluation metrics are aggregated in a dataset-balanced manner by first computing performance per dataset and then averaging across datasets to avoid bias from uneven task coverage.

2.2.7. Data Transformations

In line with prior work [1], and with the objective of diversifying data augmentation within the corpus using the implementation described in [45], transformation pipelines are generated and evaluated on the CIFAR-10 dataset. A fixed ResNet architecture is trained for a single epoch (batch size 64, learning rate 0.01, momentum 0.9, dropout 0.2), and validation accuracy is recorded. The first approach is LLM-based: prompted with the desired interface and examples of Torchvision transforms, the model proposes augmentation functions as Python code, which are syntactically validated before integration. The second approach is combinatorial: from a predefined set of Torchvision transforms, all pipelines with one, two, or three variable transforms are enumerated and extended by a fixed suffix (`Resize–ToTensor–Normalization`); variable transforms receive randomly sampled parameters. This yields 6,000 unique augmentation pipelines, each evaluated under the same training protocol.

2.3. Cross Domain Deployment and Testing

2.3.1. NN-VR

The VR-Ready Neural Network Verifier (NN-VR) provides automated validation and deployment of pre-trained LEMUR models in Unity-based virtual reality environments. Its architecture comprises (i) a Neural Network Parser that ingests ONNX exports and associated metadata from LEMUR and populates a Unity project configured

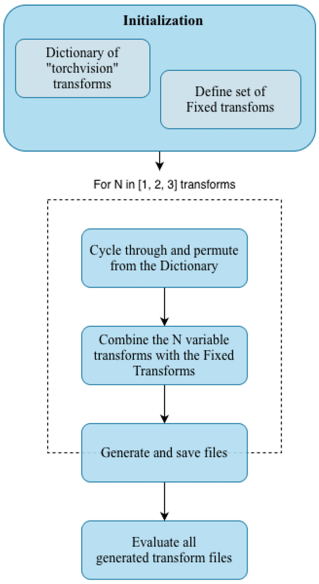

Figure 5. Overview of the brute-force generation of augmentation pipelines by permuting a fixed set of transforms and appending a standard preprocessing suffix.

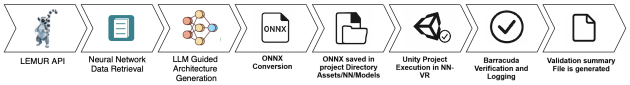

Figure 6. NN-VR pipeline

with the Barracuda inference engine; (ii) a Compatibility Verifier that imports ONNX models, checks operator coverage and shader support, and profiles GPU memory under VR-relevant settings; and (iii) an Automated Porting System that converts and optionally optimizes models, configures inference scenes, and records diagnostic logs. Performance is evaluated against VR-specific criteria: inference latency relative to an 11 ms-per-frame budget (90 Hz), additional memory overhead compared to standalone inference, and numerical consistency between Unity/Barracuda outputs and reference PyTorch predictions within a tight tolerance. The pipeline is designed for scalable application to large collections of models exported from LEMUR.

2.3.2. NN-Lite

NN-Lite [6] is an automated pipeline for deploying and benchmarking PyTorch models on the Android platform. It provides an end-to-end workflow designed to evaluate models from the dataset, managing the process from model conversion to final reporting without manual intervention.

The pipeline executes a four-stage process for each model. First, PyTorch models are automatically converted to the TensorFlow Lite (TFLite) format. This stage includes a custom wrapper to resolve the NCHW (PyTorch) to NHWC (TFLite) tensor layout disparity, ensuring compatibility and efficient memory usage. Second, the system manages the lifecycle of an Android Virtual Device (AVD), including boot-up and state monitoring. Third, a lightweight Android application is deployed to the AVD to execute model inference and collect latency metrics. Fourth, the pipeline retrieves these benchmark results and combines them with device analytics (e.g., memory, CPU architecture) into a structured JSON report.

The system is designed for large-scale, continuous operation, with built-in state management and failure recovery mechanisms. In a documented 48-hour session, the pipeline processed over 7,500 models from the LEMUR dataset. The final stage of the process is statistical consolidation, where performance metrics like task-level accuracy are aggregated with on-device latency data. This integration allows for a unified analysis of a model's software performance and its on-device execution characteristics.

3. Evaluation

All computer vision experiments are run in the AI Linux Docker image[1] on NVIDIA GeForce RTX 3090/4090 GPUs with 24 GB memory, using a Kubernetes cluster and dedicated workstations.

We performed repeated evaluations under multiple training configurations for every supported task and its associated datasets. Each evaluation run corresponds to a distinct hyperparameter setting, including batch size, number of training epochs, learning rate, momentum, dropout rate, and the selected data transformation pipeline. Every run is logged together with its accuracy and runtime, which enables systematic comparison across configurations and supports analysis of model performance variability under the same task protocol.

Evaluations use bounded training schedules to keep compute comparable while sweeping many architectures and hyperparameter settings. Metrics generally improve with additional optimization and data exposure, but convergence rates vary across configurations, so reported scores reflect compute-budgeted performance rather than uniformly converged endpoints. Accordingly, comparisons to fully trained baselines should use matched training horizons, and the results primarily quantify how efficiently each method converts compute into task performance.

4. Results

Table 2 summarizes the quantitative performance of the LEMUR 2 generative systems and deployment pipelines under bounded training budgets. Across the evaluated settings, optimization-guided generators produce the strongest and most consistent image-classification results, while prompt-based and programmatic mutation methods exhibit

[1] AI Linux: https://hub.docker.com/r/abrainone/ai-linux

higher variance in both quality and reliability. Best-per-run denotes the highest accuracy attained across all training epochs for a fixed hyperparameter configuration, allowing comparison of peak observed performance rather than final-epoch values as seen in 10, 7.

Overall, these generative mechanisms produce models that are competitive with existing manually crafted network baselines. Within groups (Fig. 9), evolutionary models attain the highest median accuracy, indicating a dense cluster of strong architectures. LLM-generated few-shot models ("alt-") occasionally match the best groups at the top but have lower median accuracy and higher variance, while programmatic AST mutations ("ast-") yield the lowest median accuracy, consistent with the sensitivity of local channel edits to the choice of input model relative to global optimization strategies.

This behavior can be explained by considering the effective size of the search space. Optimization-based generators converge faster than pure random search because the objective function provides a direction that quickly focuses the search on promising regions, especially when the coverage is moderate and the landscape is relatively smooth. As the problem becomes more complex and the search space grows, however, the landscape becomes increasingly rugged, making it harder for any method to consistently find high-quality solutions. The impact of architectural configuration, such as channel widths in the AST channel-mutated AlexNet variants show different convergence properties when compared with the original AlexNet, despite being a relatively small modification, underscoring how sensitive convergence can be to seemingly minor configurational change.

4.1. Task Extensions

Image Captioning Increasing prompt complexity (5–10 snippets) reduced runnable generations from 80% to 50%. The LLM explored diverse architectures (e.g., ConvNeXt, EfficientNet, Transformers); the best generated variant (ResNet-50 + Transformer) yielded a score of BLEU-4 = 0.317 at 50 epochs. In the longer training schedule, it can confidently compete with the baseline, (Fig. 7) approaching to the score of 0.3246. This highlights the trade-off between accuracy and architectural diversity.

Text-to-Image Methods progressed from a UNet diffusion model (CLIP 0.17–0.24) and LSTM-GAN (≈0.214) to a complex CVAE-GAN. The final CVAE-GAN, utilizing a CLIP encoder, PatchGAN discriminator, and adversarial losses, achieved the peak CLIP score of 0.2751, prioritizing semantic alignment over photorealistic textures.

Mixture-of-Experts Evaluations on CIFAR-10 showed a tuned homogeneous MoE achieving the highest accuracy (93.9%). A heterogeneous variant combining AlexNet, AirNet, DenseNet, and BagNet experts reached 93.13%, successfully outperforming all constituent backbones trained individually.

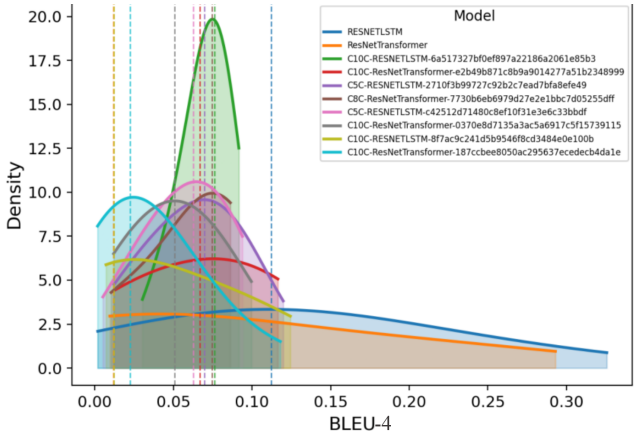

Figure 7. Best per-run image captioning BLEU-4 distributions.

4.2. Generative Systems

Genetic Algorithm A hyperparameter-restricted search yielded 62.76% test accuracy (11×11 kernel), whereas evolving block-type structures (pooling, activation, BN) improved accuracy to 80.04%. The champion model utilized batch normalization, 3×3 convolutions, and mixed pooling, demonstrating the value of block-level flexibility.

NN-RAG From 1,289 candidate blocks sourced from `timm`, `torchvision`, and `transformers`, 941 (73.0%) passed execution checks. The resulting library, tagged "rag-" in LEMUR 2, includes executable attention, convolutional, and normalization modules.

Few-Shot Prompting Results were non-monotonic: $n = 3$ exemplars achieved the highest balanced mean accuracy (53.1%) considering all datasets and improved CIFAR-100 performance by 11.6pp ($p = 0.001$). Conversely, $n = 6$ caused a 99.8% failure rate due to context overflow.

Data Transformation The best LLM-generated pipeline achieved 57.28% validation accuracy. A combinatorial search of 6,000 pipelines identified a superior configuration using a single `RandomPosterize` transform, reaching 61.24% accuracy and outperforming the generative approach.

5. Conclusion

This system was developed to produce a large and architecturally diverse collection of neural networks with extensive performance records across multiple tasks and hardware platforms, resulting in over 14,000 distinct architectures and more than 750,000 structured training records documenting model architectures, training configurations, and task-specific metrics such as accuracy. For edge devices and cross-domain coverage, NN-Lite and NN-VR automate model deployment and benchmarking while recording on-

Table 2. Quantitative summary of LEMUR 2 generative systems and deployment pipelines.

Method	Prefix	#	Dataset	Metric	Best Performance	Success Rate, %	Best / Notable Configuration
Genetic Algorithm	ga-	2000	CIFAR-10	Accuracy	0.8004	100	Block-level evolution (pooling, act., BN)
Few-Shot LLM Prompting	alt-	4033	CIFAR-10	Accuracy	0.3874[a]	—	1 exemplar (n=1) best on CIFAR-10
Fractal Networks	frac-	1258	CIFAR-10	Accuracy	0.8018[b]	97	Recursive multi-column + AMP
AST Channel Mutation	ast-	1129	CIFAR-100	Accuracy	0.3110	100	Non-standard widths + Late-stage exp.
Reinforcement Learning (GRPO)	rl-	512	CIFAR-10	Accuracy	0.6799[a]	60	Masked skeleton completion
NN-RAG	rag-	1289	CIFAR-10	Accuracy	0.9281	73.0	timm + torchvision + transformers
Data Augmentation (Brute Force)	—	6000	CIFAR-10	Accuracy	0.6124[a]	100	RandomPosterize + standard suffix
Data Augmentation (LLM-Gen)	—	280	CIFAR-10	Accuracy	0.5728[a]	22	RRC + ColorJitter + Flip + Blur
Mixture-of-Experts	moe-	8	CIFAR-10	Accuracy	0.9390	100	Homogeneous top-2 routing (MoEv7)
					0.9313	100	Heterogeneous (Alex+Air+Dense+BagNet)
Image Captioning (NN-Caption)	C*C	357	MS-COCO	BLEU-4	0.3170	>50	ResNet-50 + Transformer decoder (768-dim)
Text-to-Image	t2i-	3	—	CLIP Score	0.2751	100	CVAE-GAN + CLIP enc. + PatchGAN
NN-Lite (Android TFLite)	—	7512	—	Latency DB	—	100	NCHW→NHWC wrapper
NN-VR (Unity/Barracuda 90 Hz)	—	10244	—	Frame Time	—	95	Auto shader/memory optimization

[a] Tested on multiple datasets; the reported result corresponds to the dataset listed in the *Dataset* column. [b] Evaluated for 5 epochs.

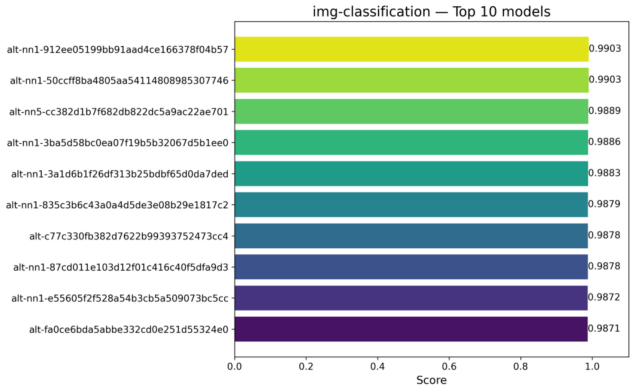

Figure 8. Top 10 image classification models across the dataset.

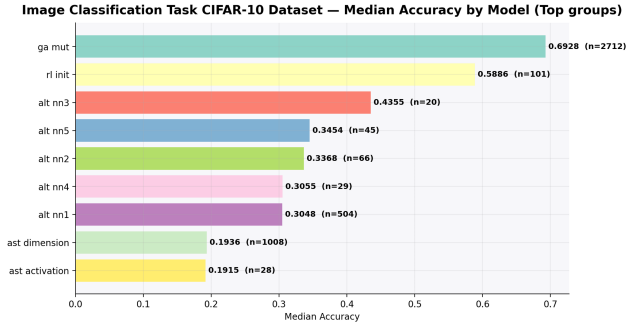

Figure 9. Median accuracy per model group for image classification task on the CIFAR-10 dataset.

device characteristics including inference latency and memory usage. By mobilizing data at scale, the framework reveals patterns that indicate which network configurations and structures are effective, supporting automated development of next-generation architectures [11]. By formalizing dependency-closed neural primitives, it also establishes a scalable methodology for "Neural Architecture Mining,"

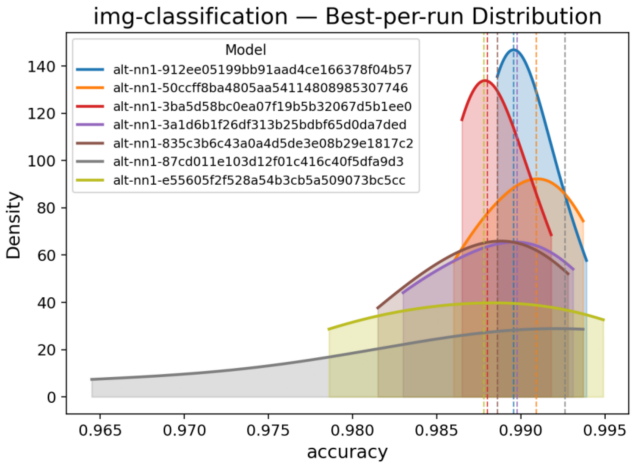

Figure 10. The distribution of the Few-Shot Architecture Prompting generated networks tested on MNIST.

transforming disparate source code into a standardized, executable substrate for automated ML workflows. Due to space constraints, further implementation details, ablation studies, and extended results are deferred to the supplementary material.

Acknowledgments. This work was partially supported by the Alexander von Humboldt Foundation.

References

[1] Nada Aboudeshish, Dmitry Ignatov, and Radu Timofte. Augmentgest: Can random data cropping augmentation boost gesture recognition performance? *arXiv preprint arXiv:2506.07216*, 2025. 5

[2] Archit Bansal, Danny Stoll, Maciej Janowski, Arber Zela, and Frank Hutter. Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. In *Proceedings of the 36th Conference on Neural Information Pro-*

cessing Systems (NeurIPS 2022) — Datasets & Benchmarks Track, 2022. Available at GitHub: https://github.com/automl/jahs_bench_201. 2, 3

[3] Wieland Brendel and Matthias Bethge. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. *arXiv preprint arXiv:1904.00760*, 2019. 4

[4] Evelyn Chee and Zhenzhou Wu. Airnet: Self-supervised affine registration for 3d medical images using neural networks. *arXiv preprint arXiv:1810.02583*, 2018. 4

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 3

[6] Saif U Din, Muhammad Ahsan Hussain, Mohsin Ikram, Dmitry Ignatov, and Radu Timofte. Ai on the edge: An automated pipeline for pytorch-to-android deployment and benchmarking. *Preprints*, 2025. 2, 6

[7] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. 2, 3

[8] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021. doi:10.1109/TPAMI.2021.3054824. 2, 3

[9] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260, 2021. 2, 3

[10] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 3

[11] Roman Kochnev et al. NNGPT: Rethinking AutoML with large language models. *arXiv preprint arXiv:2511.20333*, 2025. 2, 8

[12] Waleed Khalid et al. A retrieval-augmented generation approach to extracting algorithmic logic from neural networks. *arXiv preprint arXiv:2512.04329*, 2025. 5

[13] Waleed Khalid et al. From memorization to creativity: LLM as a designer of novel neural architectures. *arXiv preprint*, arXiv:2601.02997, 2026. 2, 5

[14] Mohamed Gado, Towhid Taliee, Muhammad Danish Memon, Dmitry Ignatov, and Radu Timofte. VIST-GPT: Ushering in the era of visual storytelling with LLMs? *arXiv preprint arXiv:2504.19267*, 2025. 2

[15] Arash Torabi Goodarzi, Roman Kochnev, Waleed Khalid, Furui Qin, Tolgay Atinc Uzun, Yashkumar Sanjaybhai Dhameliya, Yash Kanubhai Kathiriya, Zofia Antonina Bentyn, Dmitry Ignatov, and Radu Timofte. LEMUR Neural Network Dataset: Towards Seamless AutoML, 2025. 1, 2, 3

[16] Google. Tensorflow hub. https://www.tensorflow.org/hub, 2018. 2

[17] Xiaojie Gu, Dmitry Ignatov, and Radu Timofte. Resource-efficient iterative LLM-based NAS with feedback memory. *arXiv preprint*, arXiv:2603.12091, 2026. 2

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 3

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 3

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 4

[21] Krunal Jesani, Dmitry Ignatov, and Radu Timofte. LLM as a neural architect: Controlled generation of image captioning models under strict API contracts. *arXiv preprint*, arXiv:2512.14706, 2025. 2

[22] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019. 2

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 3

[24] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022. 2, 3

[25] Roman Kochnev, Arash Torabi Goodarzi, Zofia Antonina Bentyn, Dmitry Ignatov, and Radu Timofte. Optuna vs Code Llama: Are LLMs a New Paradigm for Hyperparameter Tuning? In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 5664–5674, 2025. 2

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 4

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012. 4

[28] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 4

[29] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*, 2021. 2, 3

[30] LibCST developers. LibCST: A concrete syntax tree parser and serializer library for python. https://github.com/Instagram/LibCST, 2025. Version 1.8.6, accessed November 14, 2025. 5

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 3

[32] Abhinav Mehrotra, Lukasz Dudziak, et al. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *ICLR*, 2021. 3

[33] Yash Mittal, Dmitry Ignatov, and Radu Timofte. Preparation of Fractal-Inspired Computational Architectures for Advanced Large Language Model Analysis. *arXiv preprint arXiv:2511.07329*, 2025. 2, 4

[34] Randal S. Olson and Jason H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the Workshop on Automatic Machine Learning*, pages 66–74, New York, New York, USA, 2016. PMLR. 2

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, page 311–318, USA, 2002. Association for Computational Linguistics. 3

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 2

[37] PyTorch Foundation. Pytorch hub. https://pytorch.org/hub, 2019. 2

[38] Yijian Qin, Ziwei Zhang, Xin Wang, Zeyang Zhang, and Wenwu Zhu. Nas-bench-graph: Benchmarking graph neural architecture search. *Advances in neural information processing systems*, 35:54–69, 2022. 3

[39] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 3

[40] James Reed, Zachary DeVito, Horace He, Ansley Ussery, and Jason Ansel. torch. fx: Practical program capture and transformation for deep learning in python. *Proceedings of Machine Learning and Systems*, 4:638–651, 2022. 4

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 3

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3

[43] Bhavya Rupani, Dmitry Ignatov, and Radu Timofte. Exploring the collaboration between vision models and LLMs for enhanced image classification. *Dimensions*, 27(1), 2025. doi:10.13140/RG.2.2.14615.69284. 2

[44] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 4

[45] Usha Shrestha, Dmitry Ignatov, and Radu Timofte. From brute force to semantic insight: Performance-guided data transformation design with LLMs. *arXiv preprint*, arXiv:2601.03808, 2026. 2, 5

[46] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. In *NeurIPS*, 2020. 3

[47] Tolgay Atinc Uzun, Dmitry Ignatov, and Radu Timofte. Closed-loop LLM discovery of non-standard channel priors in vision models. *arXiv preprint*, arXiv:2601.08517, 2026. 2, 4

[48] Chandini Vysyaraju, Raghuvir Duvvuri, Avi Goyal, Dmitry Ignatov, and Radu Timofte. Enhancing LLM-based neural network generation: Few-shot prompting and efficient validation for automated architecture design. *arXiv preprint*, arXiv:2512.24120, 2025. 2

[49] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 3

[50] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pages 7105–7114. PMLR, 2019. 2, 3

[51] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *ICLR*, 2020. 3

[52] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019. 3