Scientific Data

https://doi.org/10.1038/s41597-026-07893-x

Article in Press

A dataset of neural network architectures generated via large language models

Received: 9 January 2026

Accepted: 9 July 2026

Published online: 22 July 2026

Cite this article as: Daoudi, N., Cabot, J. A dataset of neural network architectures generated via large language models. *Sci Data* (2026). https://doi.org/10.1038/s41597-026-07893-x

Nadia Daoudi & Jordi Cabot

We are providing an unedited version of this manuscript to give early access to its findings. Before final publication, the manuscript will undergo further editing. Please note there may be errors present which affect the content, and all legal disclaimers apply.

If this paper is publishing under a Transparent Peer Review model then Peer Review reports will publish with the final article.

ARTICLE IN PRESS

© The Author(s) 2026. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

ARTICLE IN PRESS

A dataset of neural network architectures generated via large language models

Nadia Daoudi*
Luxembourg Institute of Science and Technology, Luxembourg
nadia.daoudi@list.lu

Jordi Cabot
Luxembourg Institute of Science and Technology, Luxembourg
University of Luxembourg, Luxembourg
jordi.cabot@list.lu

ARTICLE IN PRESS

Abstract

Neural networks are increasingly used to support decision-making. To verify their reliability and adaptability, researchers and practitioners have proposed a variety of tools and methods for tasks such as NN code verification, refactoring, and migration. These tools play a crucial role in guaranteeing both the correctness and maintainability of neural network architectures, helping to prevent implementation errors, simplify model updates, and ensure that complex networks can be reliably extended and reused. Yet, assessing their effectiveness remains challenging due to the lack of publicly available diverse datasets of neural networks that would allow systematic evaluation. To address this gap, we leverage large language models (LLMs) to automatically generate a dataset of neural networks that can serve as a benchmark for validation. The dataset is designed to cover diverse layer types and to handle multiple input data types and tasks. In total, 608 samples are generated, each conforming to a set of precise design choices. To further ensure their consistency, we validate the correctness of the generated networks using static analysis and symbolic tracing. We make the dataset publicly available to support the community in advancing research on neural network reliability and adaptability.

Background & Summary

Neural networks (NNs) have become an integral component of modern software systems, enabling them to perform complex prediction and decision-

ARTICLE IN PRESS

making tasks. Their widespread adoption spans various domains and applications, and they are increasingly at the core of software development. To support the design and development of NNs, a variety of tools have been proposed to ensure their reliability and adaptability. Verification has become a key research area, ensuring that models behave as expected and satisfy critical properties of safety or robustness [17]. In parallel, refactoring techniques have emerged to enhance models efficiency and structure without severely altering their behaviour [34]. Given the rapid evolution of NN technologies, several tools have been developed to facilitate NN migration either by transferring trained, ready-to-use models or by migrating the NN code itself across frameworks [18, 6].

Existing NN-related datasets are primarily designed for architecture benchmarking or to support NN training by providing samples for prediction tasks [13, 20]. These datasets are therefore not suitable for thoroughly evaluating tools that target NN verification, refactoring, and migration, which require a diverse set of NN architectures covering different designs, layers, tasks, and complexity levels. Evaluating such tools on a few architectures does not allow a comprehensive assessment as it does not expose the full range of NN designs they may encounter in practice. To properly evaluate the robustness and generality of NN tools, a diverse dataset of architectures is needed. However, manually collecting or designing such models is time-consuming and does not guarantee representativeness. This limitation slows progress and hinders fair and consistent evaluation across NN tools.

Large language models (LLMs) have revolutionized various domains, ranging from natural language understanding to software development and scientific computing [30]. Besides their traditional applications, LLMs have shown great potential in generating synthetic datasets. For example, they have been used to generate tabular data samples to support NN training [35]. Moreover, LLMs have shown promising results in code generation, producing functional code based on detailed user instructions [3]. Given these capabilities, LLMs stand as a promising tool for creating diverse and high-quality code-related datasets that could help overcome existing limitations in evaluation and data availability.

In this paper, we propose a dataset of NN architectures generated using LLMs. Specifically, we rely on GPT-5 [23] to create a diverse collection of NN models designed to support research on NN reliability and adaptability. Carefully crafted instructions were used to guide the generation to ensure diversity in architecture types, tasks, input types, and complexity levels. The generated NNs can handle various input data types and perform a range of prediction tasks. Moreover, they are designed with different complexity levels

ARTICLE IN PRESS

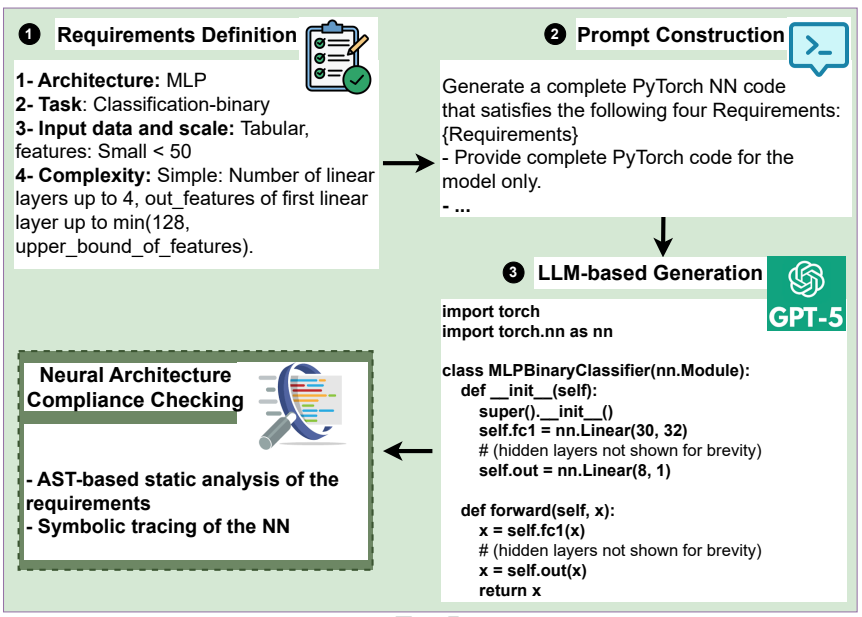


Figure 1: Overview of our LLM-driven NN dataset generation

to better reflect the variability of design that generally exist in NN models. In total, 608 NNs were generated, representing, to the best of our knowledge, the first dataset of NN architectures created to facilitate research on NN reliability and adaptability. We also perform static analysis and symbolic tracing to ensure the validity of the architectures and that they adhere to the instructions in the prompt. We make our dataset publicly available at Zenodo [24] to support the community and encourage further research on NN development.

Methods

Our NN dataset [24] was created in three main steps: Requirements Definition, Prompt Construction, and LLM-based Generation, which are illustrated in Fig. 1.

Requirements Definition

The first step in the process is to define the characteristics of the NN architectures to be generated. Since LLMs need specific instructions to guide the

ARTICLE IN PRESS

generation, it is important to feed them with precise requirements for the desired NN architectures. Specifically, we define four categories of requirements to ensure diversity:

Architecture

It refers to the type of layers used in the NN definition. Seven categories are considered: *MLP*, *CNN-1D*, *CNN-2D*, *CNN-3D*, *RNN-Simple*, *RNN-LSTM*, and *RNN-GRU*. These categories are chosen to guarantee that our dataset is diverse and covers the fundamental layers in NNs. We note that each of these architecture types is characterised by the use of a specific NN layer, hereafter referred to as a *characterising layer (CL)*, as presented in the first two columns of Table 1.

Table 1: CL and complexity thresholds of NN architectures

Architecture		**Complexity**		
Type	**CL**	**Width Thr**	**Width Variable**	**Depth Thr**
MLP	Linear	$min(128, U)$	$out_features$	4
CNN-1D	Conv1d	$min(128, U)$	$out_channels$	4
CNN-2D	Conv2d	$min(8, W//8)$	$out_channels$	4
CNN-3D	Conv3d	$min(8, W//8)$	$out_channels$	4
RNN-Simple	RNN	$min(128, U)$	$hidden_size$	2
RNN-LSTM	LSTM	$min(128, U)$	$hidden_size$	2
RNN-GRU	GRU	$min(128, U)$	$hidden_size$	2

Thr: Threshold; U : $upper_bound_features$; W : $image_width$

Task

It refers to the prediction task performed by the NN. We considered four main tasks to make the NNs diverse: *Binary-classification*, *Multiclass-classification*, *Regression* and *Representation-learning*. We note that we excluded *Multilabel-classification* as it closely resembles *Binary-classification*: both use a sigmoid output (explicitly or via losses like *BCEWithLogitsLoss*), differing mainly in the number of output neurons. Hence, including it would not contribute to the diversity of the dataset.

ARTICLE IN PRESS

Input type and scale

NNs can learn from a variety of input data types, each having unique characteristics and scales. These differences affect the learning process and determine which architectural choices are most effective for capturing the underlying structure of the data. Besides input type, we have also selected specific scales or ranges for input to drive the generation of NNs with diverse architectures. Changes in the dimensionality of input data affects how layers are structured and connected, and can also influence the inclusion of components such as normalization or activation layers, resulting in NNs with varied layer composition. We considered four input data types, *Tabular*, *Time Series*, *Text*, and *Image*, each with its corresponding input scale. These types and scales are summarized in columns 1 and 3 of Table 2.

Complexity

It influences the number of layers and the width of NNs, and is used to guide the generation of networks ranging from simple to more advanced structures. The complexity has been tailored to each architecture type, reflecting its distinct requirements for input data and intended tasks. We define two thresholds: one for the number of *CLs* and one for the width of the first CL.

Table 1 summarizes these width and depth thresholds, along with the width variable, for the different architecture types. In total, four complexity levels were considered: (1) *Simple*, referring to an NN with a basic architecture, where the number of *CLs* and the width of the first *CL* (c.f., Table 1) are expected to be below the thresholds; (2) *Wide*, an NN where the width of the first *CL* is at least the width threshold; (3) *Deep*, an NN whose total number of *CLs* is at least the depth threshold; and (4) *Wide-Deep*, an NN whose number of *CLs* and width of the first *CL* are at least the respective thresholds. For example, in a *simple MLP*, the number of *Linear* layers does not exceed four and the first Linear layer contains up to *min(128, upper_bound_of_features)* output neurons. Here, *upper_bound_of_features* is either 50 or 2000, as defined by *Input type and scale* requirement of tabular data.

For *CNN-2D* and *CNN-3D*, the width of the first *CL* is defined as $min(8, W//8)$, where W is the image width. The value of 8 was chosen to ensure that *simple* NNs remain relatively small even for large images, while *wide* NNs still exceed this baseline for smaller images. For *MLP*, *CNN-1D*, and *RNNs* (i.e., *RNN-Simple*, *RNN-LSTM* and *RNN-GRU*) architectures, the first *CL* width is defined as $min(128, U)$, with U the number of input features. 128 was cho-

ARTICLE IN PRESS

sen as a round, sufficiently large value to allow *wide* NNs to scale with the input while keeping *simple* NNs relatively small. These specific values were selected to systematically preserve the relative distinction between *simple* and *wide* NNs across varying input sizes.

For *Text* input type, *CNN-1D* and *RNNs* architectures can be used. However, since the input is text, the network starts with an *Embedding* layer rather than the standard *CL* of the architecture. The width threshold is therefore applied to this *Embedding* layer. Specifically, the width variable corresponds to *embedding_dim* of the *Embedding* layer, with a threshold of $min(128, upper_bound_of_vocab_size)$.

To avoid ambiguity in complexity types, we add the precise definition to the complexity level based on both the architecture type and the input type and scale. An example of the full complexity definition is shown in Step 1 of Fig. 1.

Prompt Construction

Listing 1: LLM prompt template

```
Generate a complete PyTorch NN code that satisfies the following
four Requirements:

Architecture: architecture
Learning Task: task
Input Type + Scale: input
Complexity: complexity

- Provide complete PyTorch code for the model only.
- All layer parameters must be fixed values inside the network,
with no variables or external constants used.
- Avoid creating excessively large layers that would cause
unreasonable memory usage.
- The model must only use standard layers directly, without
defining helper functions or custom block classes.
- Output only the code, with no comments.
```

Generating NN code with an LLM requires a carefully crafted prompt. We base our instructions on the requirements outlined in the previous section, providing details about the network architecture, input type and scale, prediction task and complexity. This approach ensures that the generated NN code follows the specified requirements, producing a variety of NNs that

ARTICLE IN PRESS

collectively cover a wide range of architectural characteristics and input data configurations. We present in Listing 1 our prompt template.

In the template, *Architecture*, *Learning Task*, *Input Type + Scale*, and *Complexity* are variables that change with each generation. We also provide additional instructions to the LLM, such as ensuring that the generated code is fully self-contained, avoiding the use of parameters defined outside the network, which could otherwise introduce unnecessary dependencies. We further avoid introducing custom block classes or helper functions, ensuring the generated networks are expressed in the standard way supported by the framework, keeping the dataset structured and consistent.

Table 2: Combinations of NN requirements considered for dataset generation

	Possible combinations			
Input Type	**Arch**	**Input Scale**	**T x C**	**Total**
Tabular	MLP	$f < 50, f > 2000$ (2 scales)	4 x 4	32
Time series	CNN–1D, RNN-Simple, RNN-LSTM, RNN-GRU	$s < 50$ & $f < 50$, $s < 50$ & $f > 2000$, $s > 2000$ & $f < 50$, $s > 2000$ & $f > 2000$ (4 scales)	4 x 4	256
Text	CNN–1D, RNN-Simple, RNN-LSTM, RNN-GRU	$s < 50$ & $v < 1k$, $s < 50$ & $v > 100k$, $s > 2000$ & $v < 1k$, $s > 2000$ & $v > 100k$ (4 scales)	4 x 4	256
Image	CNN-2D, CNN-3D	$r < 64, r > 1024$ (2 scales)	4 x 4	64

Arch: Architecture; T: Task; C: Complexity
f: $features$; s: seq_length; v: $vocab_size$; r: $resolution$

LLM-based Generation

We employ *GPT-5* [23] to produce the NN code. We use the requirements defined in Requirements Definition section to populate the prompt template presented in Prompt Construction section. To that end, we first determine the possible combinations of requirements as some combinations are irrelevant or do not correspond to valid NN architectures. For example, using

ARTICLE IN PRESS

an *MLP* to process *image* inputs is not a typical design choice, as it does not leverage the spatial structure of the data. We present a summary of the possible combinations of requirements in Table 2. In total, our dataset contains 608 NNs generated by *GPT-5*.

Data Record

The dataset is publicly available on Zenodo [24] under a CC BY 4.0 license. It consists of 608 Python files, each representing a single neural network architecture. The files are stored in a flat directory with no subdirectories.

File Naming Convention

Each file in the dataset follows the format: *architecture_ task_ input-type-scale_ complexity.py*. Each component takes one of the following values:

- architecture: one of seven types: MLP, CNN-1D, CNN-2D, CNN-3D, RNN-Simple, RNN-LSTM, RNN-GRU, corresponding to the characterising layers Linear, Conv1d, Conv2d, Conv3d, RNN, LSTM, and GRU respectively.
- task: one of four prediction tasks: binary-classification, multiclass-classification, regression, representation-learning.
- input-type-scale: combines the input data type, one of tabular, time series, text, or image, with its scale, small or large, indicating the scale of *features*, *sequence length*, *vocabulary size*, or *resolution* depending on the input type.
- complexity: one of four levels: *simple*, *wide*, *deep*, *wide-deep*, reflecting the depth and width of the network.

For example, *mlp_ classification-binary_ tabular-large_ simple.py* refers to a simple MLP for binary classification on large tabular input with a simple complexity level.

File Contents

Each .py file contains a comment at the top describing the exact prompt used to generate the network, followed by a single Python class inheriting from `torch.nn.Module` and defining the neural network, including the forward() method defining the computation graph.

ARTICLE IN PRESS

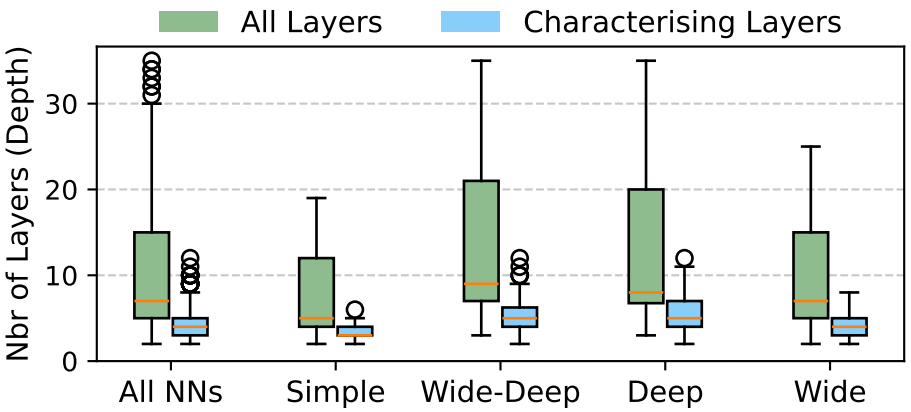


Figure 2: Depth of all layers and of *CLs* for the full NN dataset and for NNs grouped by complexity level

Data Overview

Diversity across architecture types, tasks, input types, and complexity levels is a key aspect of our dataset [24]. While the NN requirements specified for the generation define certain NN characteristics, such as the architecture type, the exact layers and tensorops used to construct the NNs are purely determined by the LLM. We therefore inspected the NN components in our dataset and found a total of 6842 layers and tensorops, comprising 38 unique types, which confirms the diversity of the structural components.

In the complexity requirements, we explicitly defined deep NNs as those with a number of *CLs* above a certain threshold. While this threshold sets the minimum number of *CLs*, the exact number of *CLs*, as well as all other layers, is determined by the LLM. We examine the distribution of NN depth in Fig. 2, which presents boxplots showing the depth of all layers and of *CLs* for both the full dataset and for NNs grouped by complexity. The overall depth of NNs ranges from 2 to 35. For NNs grouped by complexity, the patterns are consistent: *Simple* NNs generally have smaller depth (both for all layers and *CLs*), followed by *Wide*, *Deep*, and *Wide-Deep* NNs. For *Wide* NNs, no depth threshold was set; only the width was constrained. Nevertheless, the LLM produced slightly deeper NNs compared to *Simple* NNs, which could be explained by the fact that *Wide* NNs require more layers to perform feature transformations. Overall, the NN dataset contains

ARTICLE IN PRESS

a variety of layers and tensorops and includes NNs of different depths.

Technical Validation

The validity and quality of our dataset was confirmed through compliance verification, functional validation, and a utility demonstration on a use case of a NN migration tool.

Compliance Verification

Each of the 608 neural network architectures was automatically verified against the four requirements defined during dataset construction: architecture type, task, input type and scale, and complexity.

To verify the architecture requirement, the presence of at least one characterising layer (CL) was confirmed for each architecture. For the task requirement, the output layer configuration was inspected to ensure it matches the expected task, with the appropriate number of output neurons for tasks such as *classification* and *regression*. For the input type and scale, the first CL was inspected to verify that its parameters match the expected specification. This includes checking parameters such as *in_features* for *tabular* data, and *in_channels* or *input_size* for *time series*. For complexity, the number of CLs and their parameters were compared against the thresholds specified in Table 1.

This verification combines static analysis using *AST* [26], which parses the NN source code to extract all layers including functional calls [27], and symbolic tracing to confirm structural integrity and correct execution under expected inputs.

To verify the correctness of the validation process itself, we selected two neural networks per architecture type and manually inspected each one against the four defined requirements to confirm their correctness. We then ran the validation process on these 14 NNs and confirmed that they were reported as compliant with the requirements. Furthermore, we manually introduced targeted errors into these NN architectures and confirmed that they were all correctly flagged as non-compliant.

In total, 8 out of 608 architectures were found non-compliant, all belonging to the RNN-LSTM and RNN-GRU categories with large *time series* input features. The common issue was the use of a *Linear* projection layer before the *CL* (*LSTM* or *GRU*), which results in breaking the sequential structure of the input data and preventing the temporal modelling. These 8

ARTICLE IN PRESS

architectures were regenerated and successfully verified, resulting in a fully compliant dataset of 608 architectures.

Functional Validation

To confirm that the generated code is executable and functionally correct, one architecture per input data type was selected and trained on standard benchmark datasets [16, 38, 25, 2]. The results showed consistent performance, confirming the correctness and usability of the NN generated architectures.

Utility Demonstration

To further demonstrate the practical utility of the dataset, we applied it to evaluate a tool for neural network migration [6]. While the tool correctly migrates standard NN architectures, the dataset's variety and structural richness exposed varied PyTorch coding patterns that the tool did not initially support. The initial execution of the tool on the dataset showed that 200 out of 608 architectures were successfully migrated. Guided by the variety of patterns found in the dataset, the migration tool was extended to support a broader range of NN designs. As a result, all 608 architectures were successfully migrated, confirming the dataset's utility as a benchmark and its ability to benefit NN tools and drive their robustness and generalisation.

Related Work

In this section, we review related works and describe the originality of our NN dataset compared to existing datasets.

Neural Network Datasets

Existing datasets related to NNs generally focus on Neural Architecture Search (NAS) [7], which aims to automate the process of NN design to achieve the highest performance. NAS-Bench-101 [37] has proposed a set of CNN architectures mapped to their performance metrics. Similarly, NAS-HPO-Bench [15] and NAS-HPO-Bench-II [10] have considered the problem of joint optimization. Recently, Younger [36] has been proposed as a dataset containing NN architectures represented as directed acyclic graphs to assist in NN design and optimization. Other traditional datasets have been also widely used to train NNs for a range of different tasks [8, 12, 9, 16, 29, 22].

ARTICLE IN PRESS

Our work differs from these approaches by providing a dataset of diverse NNs designed to meet specific requirements on architecture, task, input, and complexity. This diversity makes it suitable for NN reliability and adaptability methods such as NN verification [17, 11], refactoring [34, 14], and migration [18, 6, 5]. In contrast, the above approaches focus on NN training or architecture search, which would prioritise performance and limit the diversity of the NNs.

Mining software repositories [42] can be considered as an alternative approach to create a dataset of neural networks. However, mined NN code potentially depends on external components and custom files, making it not directly usable without significant preprocessing and cleaning effort and does not guarantee a balanced and representative coverage of the design space.

LLM-based Dataset Generation

LLMs have been used for NAS in several works [13, 4, 21, 31]. Besides, they have been leveraged for post-architecture design tasks, such as generating NN parameters [32, 33]. Synthetic data generation for NN training has also thrived thanks to LLMs that can create diverse samples across different domains [20, 1, 19, 28]. While the main focus of these approaches is performance optimization, our work uses LLMs to generate a diverse dataset of NN architectures that can be leveraged by NN verification and migration methods.

Usage Notes

The dataset is intended to be used as a benchmark for evaluating tools related to neural network reliability and adaptability, including verification, refactoring, and code migration. Each Python file contains the source code of a single NN architecture defined as a PyTorch class, forming a complete and standalone implementation.

While the dataset covers a range of architecture types, tasks, input types, and complexity levels, several limitations should be acknowledged by users. Architectural complexity is defined in the dataset by depth and width of the networks, and topological patterns such as dense connections [39] and residual connections [40] are outside the defined scope of this dataset. Multi-branch patterns are present in 40 out of 608 architectures. Additionally, the dataset is designed to cover classical architecture families including MLP, CNN, and RNN variants, and does not include Transformer-based architectures [41], which are outside the defined scope of this version of the dataset.

ARTICLE IN PRESS

Furthermore, the dataset is generated following clean, structured PyTorch conventions, with no nested submodules or hierarchical module structures, making it a controlled baseline for the systematic evaluation of NN tools. Extending the dataset with real-world NN implementations in future work would allow for a more comprehensive evaluation across a broader range of coding patterns.

Data Availability

Our NN dataset is publicly available at Zenodo [24] under a CC BY 4.0 license.

Code Availability

The code used to generate our dataset is publicly available in our repository (`https://github.com/BESSER-PEARL/LLM-Generated-NN-Dataset`).

Author Contribution

The authors of the manuscript have contributed equally to this work.

Competing interests

The authors declare no competing interests.

Funding

This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

References

[1] Alismail, A., Lanquillon, C.: A survey of llm-based methods for synthetic data generation and the rise of agentic workflows. In: Degen, H., Ntoa, S. (eds.) Artificial Intelligence in HCI. pp. 119–135. Springer Nature Switzerland, Cham (2025)

ARTICLE IN PRESS

[2] Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L., et al.: A public domain dataset for human activity recognition using smartphones. In: Esann. vol. 3, pp. 3–4 (2013)

[3] Bistarelli, S., Fiore, M., Mercanti, I., Mongiello, M.: Usage of large language model for code generation tasks: A review. SN Computer Science **6**(6), 1–16 (2025)

[4] Chen, A., Dohan, D., So, D.: Evoprompting: Language models for code-level neural architecture search. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 7787–7817. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/184c1e18d00d7752805324da48ad25be-Paper-Conference.pdf

[5] Daoudi, N., Alfonso, I., Cabot, J.: Modelling neural network models. In: Grabis, J., Vos, T.E.J., Escalona, M.J., Pastor, O. (eds.) Research Challenges in Information Science. pp. 130–139. Springer Nature Switzerland, Cham (2025), https://link.springer.com/chapter/10.1007/978-3-031-92471-2_10

[6] Daoudi, N., Alfonso, I., Cabot, J.: Towards Migrating Neural Network Implementations. In: Proceedings of the 3rd International Conference on AI Engineering (AIware 2026) (2026), preprint available at https://arxiv.org/abs/2511.02610

[7] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research **20**(55), 1–21 (2019), http://jmlr.org/papers/v20/18-598.html

[8] Gong, Y., Liu, G., Xue, Y., Li, R., Meng, L.: A survey on dataset quality in machine learning. Information and Software Technology **162**, 107268 (2023). https://doi.org/https://doi.org/10.1016/j.infsof.2023.107268, https://www.sciencedirect.com/science/article/pii/S0950584923001222

[9] Hamid, Y., Balasaraswathi, V.R., Journaux, L., Sugumaran, M.: Benchmark datasets for network intrusion detection: A review. Int. J. Netw. Secur. **20**(4), 645–654 (2018)

ARTICLE IN PRESS

ARTICLE IN PRESS

[10] Hirose, Y., Yoshinari, N., Shirakawa, S.: NAS-HPO-Bench-II: A benchmark dataset on joint optimization of convolutional neural network architecture and training hyperparameters. In: Balasubramanian, V.N., Tsang, I. (eds.) Proceedings of The 13th Asian Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 157, pp. 1349–1364. PMLR (17–19 Nov 2021), https://proceedings.mlr.press/v157/hirose21a.html

[11] Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International conference on computer aided verification. pp. 3–29. Springer (2017)

[12] Imran, Qayyum, F., Kim, D.H., Bong, S.J., Chi, S.Y., Choi, Y.H.: A survey of datasets, preprocessing, modeling mechanisms, and simulation tools based on ai for material analysis and discovery. Materials **15**(4) (2022). https://doi.org/10.3390/ma15041428, https://www.mdpi.com/1996-1944/15/4/1428

[13] Jawahar, G., Abdul-Mageed, M., Lakshmanan, L.V., Ding, D.: Llm performance predictors are good initializers for architecture search. arXiv preprint arXiv:2310.16712 (2023)

[14] Kaplunovich, A., Yesha, Y.: Refactoring of neural network models for hyperparameter optimization in serverless cloud. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. pp. 311–314 (2020)

[15] Klein, A., Hutter, F.: Tabular benchmarks for joint architecture and hyperparameter optimization. arXiv preprint arXiv:1905.04970 (2019)

[16] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)

[17] Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: Advances, challenges and perspectives. arXiv preprint arXiv:1805.09938 (2018)

[18] Liu, Y., Chen, C., Zhang, R., Qin, T., Ji, X., Lin, H., Yang, M.: Enhancing the interoperability between deep learning frameworks by model conversion. In: Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. pp. 1320–1330 (2020)

ARTICLE IN PRESS

ARTICLE IN PRESS

[19] Long, L., Wang, R., Xiao, R., Zhao, J., Ding, X., Chen, G., Wang, H.: On llms-driven synthetic data generation, curation, and evaluation: A survey. arXiv preprint arXiv:2406.15126 (2024)

[20] Nadăş, M., Dioşan, L., Tomescu, A.: Synthetic data generation using large language models: Advances in text and code. IEEE Access **13**, 134615–134633 (2025). https://doi.org/10.1109/ACCESS.2025.3589503

[21] Nasir, M.U., Earle, S., Togelius, J., James, S., Cleghorn, C.: Llmatic: Neural architecture search via large language models and quality diversity optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 1110–1118. GECCO '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3638529.3654017, `https://doi-org.proxy.bnl.lu/10.1145/3638529.3654017`

[22] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., et al.: Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. vol. 2011, p. 7. Granada (2011)

[23] OpenAI: Gpt-5 system card (2025), `https://cdn.openai.com/gpt-5-system-card.pdf`, accessed October 09, 2025

[24] Daoudi, N., Cabot, J.: NN-dataset. Zenodo (2026), https://doi.org/10.5281/zenodo.20490365

[25] Pace, R.K., Barry, R.: Sparse spatial autoregressions. Statistics & Probability Letters **33**(3), 291–297 (1997)

[26] Python Software Foundation: ast — Abstract Syntax Trees — Python 3.13.5 documentation (2025), `https://docs.python.org/3/library/ast.html`, accessed October 09, 2025

[27] PyTorch: torch.nn.functional — pytorch documentation (2025), `https://pytorch.org/docs/stable/nn.functional.html`, accessed: October 09, 2025

[28] Ronval, B., Dupont, P., Nijssen, S.: Tagal: Tabular data generation using agentic llm methods. arXiv preprint arXiv:2509.04152 (2025)

[29] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet

ARTICLE IN PRESS

large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)

[30] Sajjadi Mohammadabadi, S.M., Kara, B.C., Eyupoglu, C., Uzay, C., Tosun, M.S., Karakuş, O.: A survey of large language models: Evolution, architectures, adaptation, benchmarking, applications, challenges, and societal implications. Electronics **14**(18) (2025). https://doi.org/10.3390/electronics14183580, `https://www.mdpi.com/2079-9292/14/18/3580`

[31] Song, X., Tian, Y., Lange, R.T., Lee, C., Tang, Y., Chen, Y.: Position: Leverage foundational models for black-box optimization. arXiv preprint arXiv:2405.03547 (2024)

[32] Soro, B., Andreis, B., Chong, S., Hwang, S.J.: Instruction-guided autoregressive neural network parameter generation. In: Workshop on Neural Network Weights as a New Data Modality

[33] Wang, J., Zhang, Y., Li, X.: Neurogen: Neural network parameter generation via large language models. arXiv preprint arXiv:2505.12470 (2025)

[34] Wang, S., Hu, X., Wang, B., Yao, W., Xia, X., Wang, X.: Refactoring deep learning code: A study of practices and unsatisfied tool needs. In: International Conference on Software Maintenance and Evolution (2025)

[35] Wang, Y., Feng, D., Dai, Y., Chen, Z., Huang, J., Ananiadou, S., Xie, Q., Wang, H.: Harmonic: Harnessing llms for tabular data synthesis and privacy protection. Advances in Neural Information Processing Systems **37**, 100196–100212 (2024)

[36] Yang, Z., Gao, W., Peng, L., Huang, Y., Tang, F., Zhan, J.: Younger: The first dataset for artificial intelligence-generated neural network architecture. arXiv preprint arXiv:2406.15132 (2024)

[37] Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: NAS-bench-101: Towards reproducible neural architecture search. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7105–7114. PMLR (09–15 Jun 2019), `https://proceedings.mlr.press/v97/ying19a.html`

ARTICLE IN PRESS

ARTICLE IN PRESS

[38] Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015), https://proceedings.neurips.cc/paper_files/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf

[39] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K. Q.: Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708). (2017)

[40] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778). (2016)

[41] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems, 30. (2017)

[42] Hassan, A. E.: The road ahead for mining software repositories. In 2008 frontiers of software maintenance (pp. 48-57). IEEE. (2008, September)

ARTICLE IN PRESS