Intuitive Design of Deep Learning Models through Visual Feedback

JunYoung Choi*
VIENCE Inc.
Korea University

Sohee Park†
VIENCE Inc.

GaYeon Koh‡
Korea University

Youngseo Kim§
VIENCE Inc.

Won-Ki Jeong¶
VIENCE Inc.
Korea University

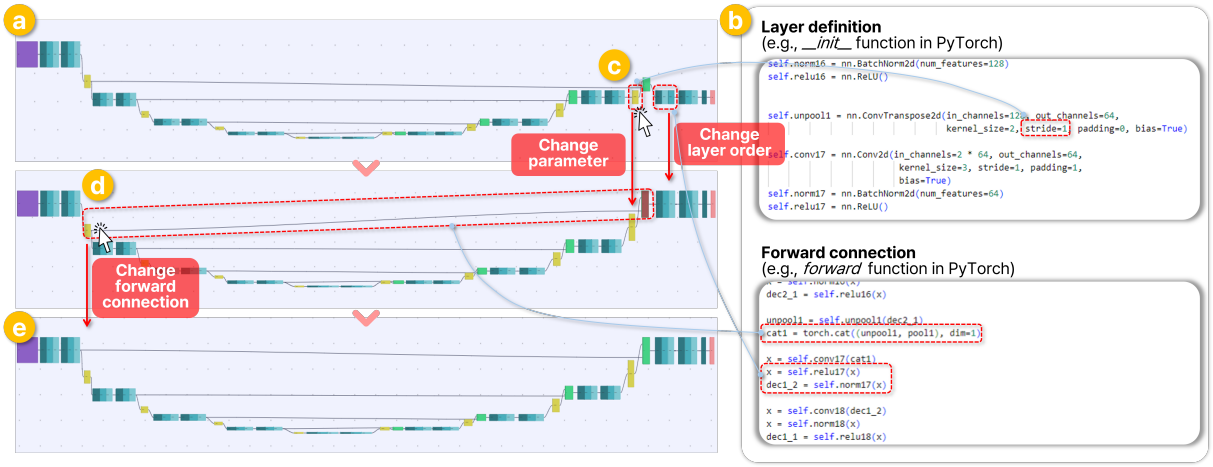


Figure 1: Proofreading structural issues in a deep learning model (U-Net [17]) can be effectively done using a proposed visual feedback-based no-code approach (a) compared to the conventional code-based method (b). Developers can identify and correct errors in the up pooling layer by comparing the spatial dimension (height of the node) of the layer outputs in the initially constructed deep learning model (c). Errors in layer placement can be identified and corrected through the color of the nodes (consecutive colors are assigned if there is a recommended layer placement order). Incorrect forward connections can be corrected through the representation of expected structural errors (e.g., a flashing red when different spatial dimension tensors are combined) (d). This iterative process of correction makes it easy to build a model that reflects the developer's intent (e).

Abstract

In the rapidly evolving field of deep learning, traditional methodologies for designing models predominantly rely on code-based frameworks. While these approaches provide flexibility, they create a significant barrier to entry for non-experts and obscure the immediate impact of architectural decisions on model performance. In response to this challenge, recent no-code approaches have been developed with the aim of enabling easy model development through graphical interfaces. However, both traditional and no-code methodologies share a common limitation that the inability to predict model outcomes or identify issues without executing the model. To address this limitation, we introduce an intuitive visual feedback-based no-code approach to visualize and analyze deep learning models during the design phase. This approach utilizes dataflow-based visual programming with dynamic visual encoding of model architecture. A user study was conducted with deep learning developers to demonstrate the effectiveness of our approach in enhancing the model design process, improving model understanding, and facilitating a more intuitive development experience. The findings of this study suggest that real-time architectural visualization significantly contributes to more efficient model development and a deeper understanding of model behaviors.

*e-mail: jychoi@vience.co.kr
†e-mail: wings159@vience.co.kr
‡e-mail: hellenkoh@gmail.com
§e-mail: k0seo0330@vience.co.kr
¶e-mail: wkjeong@vience.co.kr

Index Terms: Deep learning, visual programming, explainable AI.

1 Introduction

Deep learning represents one of the most innovative developments in recent times, having achieved breakthroughs in a variety of applications including image recognition, natural language processing, and medical diagnosis [4]. The foundation of these achievements is the design and training process of complex and diverse neural network models. Traditionally, these models have been developed using code-based frameworks [14, 13], which provide a high degree of flexibility and control to expert developers. However, this approach demands considerable effort because it involves understanding deep learning concepts and learning programming languages such as Python. As a result, most end-users, who typically lack expertise in these areas, find it challenging to build and utilize deep learning models due to these obstacles.

Recently, no-code approaches [9, 2, 21] have emerged in an effort to make deep learning model development accessible to a broader user base. This approach allows models to be constructed easily through a graphical interface, allowing even users with limited programming skills to develop complex deep learning models. This promotes the generalization of deep learning technology and paves the way for developers from diverse backgrounds to contribute to the field.

Although deep learning programming has become easier with the efforts, additional steps are still necessary to validate whether a deep learning model is designed as intended, such as executing the model at the code level. For example, structural errors such as inefficient placement of layers in the designed deep learning model, or incorrect input and output shapes, can only be discovered through the tedious and annoying debugging process of executing and ana-

lyzing the model. What if it were possible to check whether everything is going as intended during the model design phase?

To address this issue, we propose a visual feedback-based no-code approach that enables developers to intuitively understand and analyze the structure of the model by visualizing its architecture in real time throughout the model development process. The proposed method is based on a dataflow-based visual programming approach. The method includes the dynamic changing visual encoding of each node in the dataflow, the aligning of these nodes, and the representation of predicted structural errors. A user study was conducted with deep learning developers to demonstrate that the real-time architecture visualization enables the model design process more efficient, improves model understanding, and provides a more intuitive development experience. The results contribute to the advancement of deep learning technology by enabling developers to better predict model performance and behavior, thus allowing for the development of more effective models.

We have also developed a graphical user interface (GUI) applying proposed method through a web platform[1], enabling researchers to try it out in practice.

2 Related Work

2.1 Conventional deep learning development

In recent years, the field of artificial intelligence models has continued to expand, and the deep learning technology has been developed [23, 22, 18]. With the rapid development of the deep learning field, conventional deep learning model design methodologies are mainly based on code-based frameworks [16] utilizing libraries such as TensorFlow [1, 14] and PyTorch [13]. Although these tools offer developers flexibility and ease of use, enabling them to tackle complex architectures, they still require familiarity with the programming language underlying the code, such as Python, and expertise in deep learning.

2.2 No-code deep learning development

Recently, there has been several researches on the construction of deep learning models in a no-code manner, utilizing GUIs to facilitate high accessibility. Tamilselvam *et al.* [21] proposed a visual programming method, enabling users to construct deep learning models by connecting layers and convert them to code through simple interactions. Calo *et al.* [2] have developed visual programming method with simple visualizations of data passing through the layers and visualizations of training metrics to analyze the model. However, these methods are not easy to analyze and validate the model during the deep learning programming phase, such as whether the actual model was built as intended and whether there were any structural issues [3]. Several studies have also discussed the importance of this no-code approach to deep learning programming. In particular, Esposito *et al.* [3] have highlighted the importance of developing methods or platforms that allow real end-users, rather than deep learning experts, to develop the models themselves. Furthermore, Li *et al.* [9] discussed that these methods should not only facilitate rapid and iterative prototyping, but also provide proper feedback. While no-code deep learning methods are being developed and their importance is being studied, research on visualization methods for understanding and providing appropriate feedback to validate the built models is just beginning.

3 Methods

In this section, we introduce a visual feedback-based no-code approach for building deep learning models that provides appropriate feedback during the model design phase.

[1] https://vience.io/vience-canvas/mlops/sample

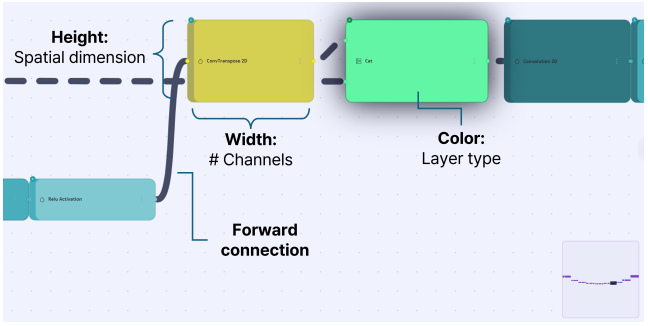
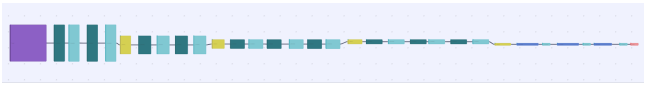
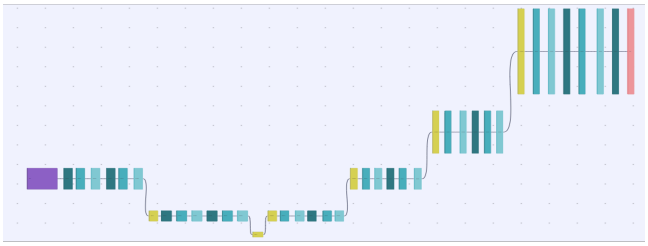


Figure 2: The visual encoding elements of each node in visual programming. Each node is visualized using information from a corresponding layer and information from its output tensor. The visualization of the node is modified in real time as the developer modifies the parameters of the layer or modifies the connections between nodes.



(a) Simple alignment



(b) U-Net shape alignment

Figure 3: Node alignment methods that utilize forward connection and layer information. Depending on the type of the model, developers can build a model using the Simple alignment method (a), which considers the topological order of the forward connections and places them in a horizontal line, or the U-Net shape alignment method (b), which additionally adjusts the vertical height of the nodes through the spatial dimension of the corresponding layer's output tensor.

Note that the scope of this work is limited to the case where the input data has a spatial dimension (e.g., an image) to better demonstrate the effectiveness of visual feedback.

3.1 Model construction with visual feedback

To support the prediction of outcomes during the design phase of a deep learning model, we utilize a dataflow-based visual programming approach that connects nodes (corresponding to each layer of a deep learning model) to create a graph [21]. We propose a dynamic visual encoding method that interactively visualizes each node through the corresponding layer's information, such as output tensor shape, layer type, and layer parameters.

Deep learning model flow: We named the deep learning model created by connecting each node through visual programming method as *model flow*. Each node corresponds to a layer of the model, such as fully-connected layer, convolution layer, activation layer, etc., and the connection between nodes corresponds to the forward connection of the model. The model flow starts from the dataset node (the starting point of the forward connection) and ends at the trainer node (the final output of the forward connection). At each node, developers can set layer parameters, such as kernel, stride, padding size, etc., according to the corresponding layer type.

Node visual encoding: Each node in the model flow has its vi-

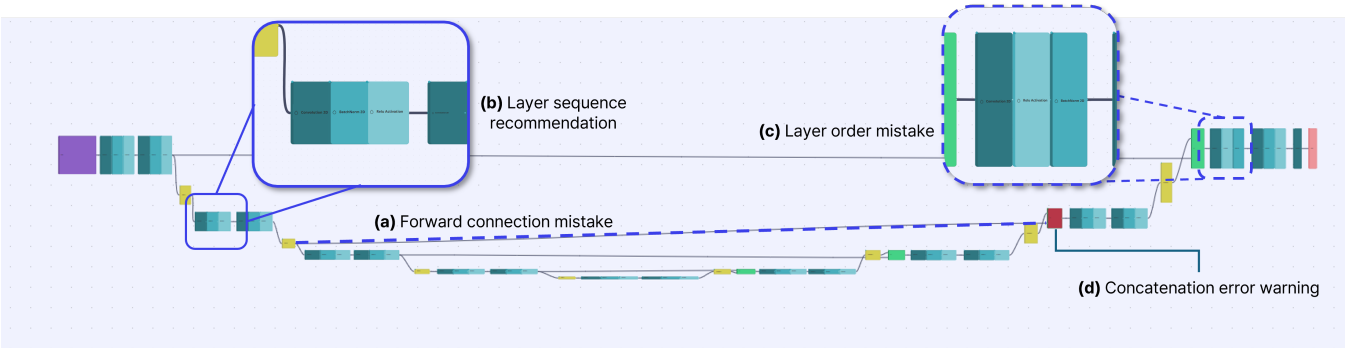


Figure 4: Error cases that can be detected through visual feedback in models from the no-code method. The connections between nodes and the node alignment method can be used to find incorrectly connected forword connections (a), and tensors with different spatial dimensions connected to a concatenation node will blink red to indicate possible errors (b). Furthermore, if there is a recommended layer placement order, the nodes are displayed in a continuous color (c) to help developers find out where the layer placement order is incorrect (d).

sual encoding change in real time according to the information in the corresponding layer (Fig. 2). The visual encoding method is utilized from the two-dimensional visualization method used in many existing deep learning papers [17, 6]. The horizontal width of a node is proportional to the number of channels (or features) in the layer output tensor, and the vertical height is proportional to the spatial dimension of the layer output tensor.

Node alignment: The nodes are aligned from left to right according to the order of their forwarding connection. In the event of concatenation, such as skip connections, they are placed in topological order. Node alignment in the vertical direction can be accomplished through the use of a *simple alignment* method (Fig. 3a) that positions nodes according to model characteristics (e.g., in the form of a model figure of a convolutional neural network (CNN) model [10]) or a *U-Net shape alignment* method (Fig. 3b) that aligns nodes according to their spatial dimension (e.g., in the form of a model figure of an encoder-decoder model [17]). In the *U-Net shape alignment* method, nodes with the same spatial dimension outputs are aligned in the same row.

Structural error representation: The above node alignment methods allow developers to compare the output sizes of layers in the same row, or to visually detect and correct errors in setting layer-specific parameters (Fig. 4). Furthermore, if there is a correct sequence for the layers of a deep learning model (e.g., the order of convolution, normalization, and activation layers proposed in Ioffe *et al.*'s study [7]), the color brightness is changed step by step according to the recommended sequence (Fig. 4b), thereby enabling developers to visually verify the proper positioning of the layers (Fig. 4c). In the event that structural issues are expected, such as the concatenation of tensors with different spatial dimensions through concatenation nodes, a visual warning (Fig. 4d) is displayed to detect and correct structural errors during the deep learning model design phase.

3.2 Implementation

The proposed method operates through a web-based GUI, and the process of generating deep learning code and training from the created model flow is performed on the server side by connecting to the server through a RESTful API.

To build the web-based GUI, we used the TypeScript-based React [26] and the visual programming library Rete.js [19]. For the RESTful API, we used FastAPI [15] in Python environment. The deep learning code converted from the model flow is based on the PyTorch library [13].

4 Evaluations

A user study was conducted with five deep learning developers (**P1-5**) to investigate the potential of visual feedback in the process of building deep learning models. The participants had at least one year of experience in the field of deep learning development, primarily using Visual Studio Code [11] and PyTorch [13]-based development. During the pre-interviews, they identified a number of challenges associated with the conventional approach to working, including the necessity of possessing a certain level of expertise to implement a desired idea and the need to modify others' code due to incompatibilities in format. One participant (**P5**) also discussed the difficulty of understanding the model structure. He also mentioned that he has been using tools such as TorchViz [25] and TensorFlow Graph Visualizer [24] to visualize the model structure, but they are limited in their ability to provide comprehensive insights (e.g., only show layer information).

4.1 Task and procedure

The study involved three tasks: error detection (**T1**), structural difference identification (**T2**), and performance improvement (**T3**). Each task was executed using three different methods for building deep learning models: the conventional method (code-based programming, **M1**), the basic visual programming-based method without visual feedback [21] (**M2**), and the visual programming-based method with visual feedback (our approach, **M3**). We used the PyTorch library in the Visual Studio Code environment for **M1**.

The experiment included a pre-interview, task executions, and a post-interview. Participants performed the tasks sequentially (**T1** $\rightarrow$ **T2** $\rightarrow$ **T3**) with varying method orders (**P1-2: M1** $\rightarrow$ **M2** $\rightarrow$ **M3, P3-5: M3** $\rightarrow$ **M2** $\rightarrow$ **M1**). In addition, dummy tasks were included to minimize the influence of previous tasks. The post-interview measured System Usability Scale (SUS) scores [8] of the proposed method and gathered feedback on the participants' experiences.

T1: Error detection Participants identified structural errors in two given deep learning models using each method. Each given model contained two errors about the placement of layers (Fig. 4b, c) and two errors about forward connections (Fig. 4a, d). The purpose of this task was to determine how quickly and accurately each method could find errors in deep learning models. Each time participants found an error, they described the error to an interviewer, who recorded the time and accuracy of the error detection.

T2: Structural difference identification Participants identified differences between illustrations of two popular deep learning models (VGG16 [10] and U-Net [17]) and the given models. The given models had missing layers, different parameter values (output channels, padding, etc.), or different layer orders. The first given model

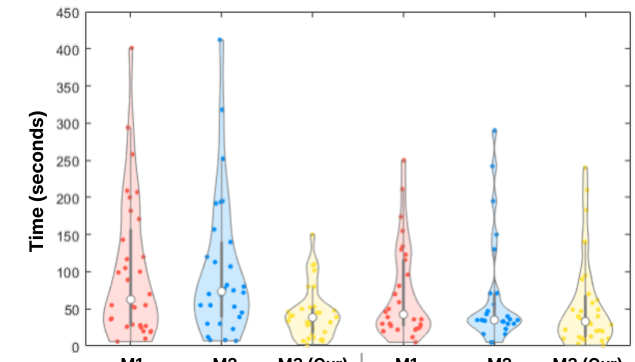


Figure 5: Results for **T1** and **T2**. The plot presents the time taken to identify each structural error or each structural difference. The table presents the percentage of correctly identified errors or differences among the total number of structural errors or total structural differences (accuracy).

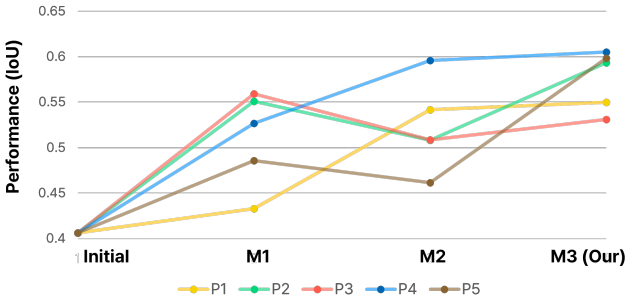


Figure 6: Results for **T3**. The plot presents the highest performance (IoU) improvement observed with each method during **T3**.

(VGG16) had a total of three differences, and the second given model (U-Net) had a total of six differences. The experiment assessed how quickly participants could understand and analyze the architecture of the models. The time and accuracy of each identification were recorded.

T3: Performance improvement Participants optimized an initial model's performance within 10 minutes. The initial model was a simplified version of the U-Net model for binary segmentation, and its performance was evaluated based on the Intersection over Union (IoU) metric at 20 epochs. For **M1**, model training was conducted on an Ubuntu server equipped with an Intel(R) Xeon(R) Gold 6326 CPU and a NVIDIA RTX A6000 GPU. **M2** and **M3** also utilized the same server via RESTful API for the training process. The training dataset utilized in the experiments consisted of a sampling of 213 images (training: 191, validation: 22) from the Lizard dataset [5]. For **M1**, the participants were provided with all the fundamental code required to train and validate an initial deep learning model, including PyTorch model code, training code, etc. The purpose of this task was to study the efficacy of each method in providing insight into the model and improving its performance in a relatively short period of time. The interviewer observed the participants' behavior while they performed the task.

4.2 Results

Fig. 5 depicts the results for **T1** and **T2**. In **T1**, the accuracy (ratio of the number of identified errors) was 80% for **M1**, 75% for **M2**, and 80% for **M3**. The mean time to identify each error was 102 seconds (±17) for **M1**, 103 seconds (±17) for **M2**, and 44 seconds (±6) for **M3**. In **T2**, the accuracy (ratio of the number of identified differences) was 67% for **M1**, 58% for **M2**, and 78% for **M3**. The mean time to identify each structural difference was 72 seconds (±12) for **M1**, 67 seconds (±14) for **M2**, and 55 seconds (±10) for **M3**. Consequently, the proposed method achieved a significantly shorter average time in **T1** compared to other methods (adjusted p-values are 0.016 and 0.014 for **M1** and **M2**, respectively, from a multiple comparison test by Tukey's HSD [12]). Furthermore, the proposed method achieved a significantly higher accuracy than other methods in **T2**. These results indicate that visual feedback can facilitate the validation of deep learning models.

Fig. 6 shows the performance (IoU) improvement of the deep learning model with each method in **T3**. The performance of a given initial deep learning model was 0.40, and the average performance improvement with each method was 0.51 (±0.02) for **M1**, 0.52 (±0.02) for **M2**, and 0.58 (±0.01) for **M3**. A notable result here is that **P1** was able to improve the model performance much faster with **M3**, as it took 5.83 minutes with **M2** and 3.83 minutes with **M3**, although the model performance obtained with **M2** and **M3** are not significantly different (0.54 and 0.55, respectively). Furthermore, it was observed that **P2-5** encountered difficulties in improving model performance due to structural errors when adding multiple layers to the model with **M1** and **M2**. In contrast, with **M3**, it was observed that errors were easily corrected even when they occurred. For example, **P5** encountered a structural error while connecting multiple layer nodes to the model through **M2** and was unable to fix the error within the 10-minute time limit. However, after moving the model to the **M3**, the structural error was found and fixed within 12 seconds.

The average SUS score of our proposed method is 83 (±2.3), which indicates excellent usability.

4.3 Feedback

Participants noted that the proposed method's ability to automatically align multiple layers significantly facilitated the rapid identification of problematic points during model debugging. They also highlighted the intuitive and easy-to-use UI that can be used immediately without any separate training as a major advantage.

However, some participants noted that the implementation is still limited to basic layer types, which are insufficient for developing complex modern models. Additionally, while the dynamic visualization of nodes plays a significant role in understanding the model, as the model complexity increases and the number of nodes grows, controlling the model becomes more challenging.

5 Conclusion

We proposed a visual programming approach enhanced by visual feedback for the development of deep learning models. Our user study demonstrated that the interpretation of deep learning models becomes faster and more accurate with the proper visual feedback. Although the advancement of no-code techniques is broadening the accessibility of deep learning, the importance of proper visualization for understanding these models is often overlooked. Our findings emphasize that the development of appropriate visual feedback is as crucial as the development of no-code deep learning techniques.

In the future, we plan to apply the proposed method to develop a visualization-based MLOps [20] framework that can design, train, validate, and deploy deep learning models with visual feedback. We will also study visualization methods for model abstraction to enable easy manipulation of complex models through GUI.

ACKNOWLEDGMENTS

This work was supported by the Startup growth technology development program (RS-2023-00257786) funded by the Ministry of SMEs and Startups (MSS, Korea), ICT Creative Consilience Program (RS-2020-II201819) through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT), and Basic Science Research Program (NRF-2021R1A6A1A13044830) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education.

REFERENCES

[1] M. Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, pp. 1–1, 2016. 2

[2] T. Calò and L. De Russis. Towards a visual programming tool to create deep learning models. In *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pp. 38–44, 2023. 1, 2

[3] A. Esposito, M. Calvano, A. Curci, G. Desolda, R. Lanzilotti, C. Lorusso, and A. Piccinno. End-user development for artificial intelligence: A systematic literature review. In *International Symposium on End User Development*, pp. 19–34. Springer, 2023. 2

[4] M. Gheisari, F. Ebrahimzadeh, M. Rahimi, M. Moazzamigodarzi, Y. Liu, P. K. Dutta Pramanik, M. A. Heravi, A. Mehbodniya, M. Ghaderzadeh, M. R. Feylizadeh, et al. Deep learning: Applications, architectures, models, tools, and frameworks: A comprehensive survey. *CAAI Transactions on Intelligence Technology*, 8(3):581–606, 2023. 1

[5] S. Graham, M. Jahanifar, A. Azam, M. Nimir, Y.-W. Tsang, K. Dodd, E. Hero, H. Sahota, A. Tank, K. Benes, et al. Lizard: a large-scale dataset for colonic nuclear instance segmentation and classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 684–693, 2021. 4

[6] B. He, X. Yang, H. Wang, Z. Wu, H. Chen, S. Huang, Y. Ren, S.-N. Lim, and A. Shrivastava. Towards scalable neural representation for diverse videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6132–6142, 2023. 3

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015. 3

[8] J. R. Lewis. The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018. 3

[9] L. Li and Z. Wu. How can no/low code platforms help end-users develop ml applications?-a systematic review. In *International Conference on Human-Computer Interaction*, pp. 338–356. Springer, 2022. 1, 2

[10] S. Mascarenhas and M. Agarwal. A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification. In *2021 International conference on disruptive technologies for multi-disciplinary research and applications (CENTCON)*, vol. 1, pp. 96–99. IEEE, 2021. 3

[11] Microsoft Corporation. Visual Studio Code. https://code.visualstudio.com/. 3

[12] A. Nanda, B. B. Mohapatra, A. P. K. Mahapatra, A. P. K. Mahapatra, and A. P. K. Mahapatra. Multiple comparison test by tukey's honestly significant difference (hsd): Do the confident level control type i error. *International Journal of Statistics and Applied Mathematics*, 6(1):59–65, 2021. 4

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1, 2, 3

[14] S. Pattanayak. *Pro deep learning with TensorFlow 2.0: A mathematical approach to advanced artificial intelligence in Python*. Springer, 2023. 1, 2

[15] S. Ramírez. FastAPI. https://fastapi.tiangolo.com/. 3

[16] S. Raschka, Y. H. Liu, V. Mirjalili, and D. Dzhulgakov. *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt Publishing Ltd, 2022. 2

[17] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241. Springer, 2015. 1, 3

[18] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed. A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru. *arXiv preprint arXiv:2305.17473*, 2023. 2

[19] V. Stoliarov. Rete.js. The JavaScript framework for visual programming. https://rete.js.org/. 3

[20] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas. Mlops-definitions, tools and challenges. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0453–0460. IEEE, 2022. 4

[21] S. G. Tamilselvam, N. Panwar, S. Khare, R. Aralikatte, A. Sankaran, and S. Mani. A visual programming paradigm for abstract deep learning model development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction*, pp. 1–11, 2019. 1, 2, 3

[22] R. Thirunavukarasu, R. Gnanasambandan, M. Gopikrishnan, V. Palanisamy, et al. Towards computational solutions for precision medicine based big data healthcare system using deep learning models: A review. *Computers in Biology and Medicine*, 149:106020, 2022. 2

[23] M. Tsuneki. Deep learning models in medical image analysis. *Journal of Oral Biosciences*, 64(3):312–320, 2022. 2

[24] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2017. 3

[25] S. Zagoruyko. Pytorchviz: A small package to create visualizations of pytorch execution graphs and traces, 2018. 3

[26] F. Zammetti. *Modern Full-Stack Development: Using TypeScript, React, Node. js, Webpack, and Docker*. Springer, 2020. 3