DeepFlow: A Flow-Based Visual Programming Tool for Deep Learning Development

Tommaso Calò
Dipartimento di Automatica e Informatica
Politecnico Di Torino
Torino, Torino, Italy
tommaso.calo@polito.it

Luigi De Russis
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
luigi.derussis@polito.it

Abstract

Visual programming tools have recently been introduced to enable Deep Learning (DL) development without the need for expertise in traditional programming languages and frameworks. However, these tools often exhibit limitations in scalability for complex architectures and lack real-time debugging capabilities. This paper introduces DeepFlow, a flow-based visual programming tool (VPT) designed to address these challenges by leveraging the inherently visual nature of DL models as sequences of learnable functions. DeepFlow incorporates hierarchical abstraction mechanisms through "supernodes" to support model scalability, which is crucial for modern, complex architectures. Additionally, it introduces interactive debugging in the model design phase, allowing developers to validate network architectures before execution. We conducted a user study with 16 DL developers, involving typical DL model design tasks. We assessed DeepFlow using quantitative usability metrics, and post-task interviews to evaluate user perceptions and workflow integration across different expertise levels. Results demonstrated high usability and user satisfaction, and highlighted DeepFlow's effectiveness for rapid model iteration and as a learning aid for complex DL architectures, while also identifying areas for improvement.

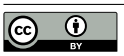
CCS Concepts

• **General and reference** → **Empirical studies**; • **Human-centered computing** → **Human computer interaction (HCI)**; *Interactive systems and tools*; Empirical studies in HCI; • **Computing methodologies** → *Machine learning*; *Artificial intelligence.*

Keywords

Visual Programming Tools, Deep Learning, Neural Network Design, Interactive Debugging, Model Scalability, User Experience, Rapid Prototyping, Hierarchical Abstraction, Human-Computer Interaction

ACM Reference Format:
Tommaso Calò and Luigi De Russis. 2025. DeepFlow: A Flow-Based Visual Programming Tool for Deep Learning Development. In *30th International Conference on Intelligent User Interfaces (IUI '25), March 24–27, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3708359.3712109



IUI '25, Cagliari, Italy

ACM ISBN 979-8-4007-1306-4/25/03
https://doi.org/10.1145/3708359.3712109

1 Introduction

Rapid progress in Deep Learning (DL) has catalyzed significant advances across numerous fields, including image recognition, natural language processing, autonomous driving, medical diagnosis, and drug discovery. As DL applications expand beyond traditional computer science domains into practical, domain-specific implementations, there is a growing need to make these technologies accessible to a broader audience of domain experts. Empowering these professionals to directly engage with DL model development can accelerate research and increase productivity in their specialized areas. However, current popular frameworks for DL development — such as PyTorch [37], TensorFlow [1], and Keras [9] — rely on programming languages. This reliance poses significant challenges [42] for non-computer science experts attempting to leverage DL methods. Even within the computer science community, there is a common perception that existing DL frameworks lack essential features for quicker and more efficient implementation and prototyping [42].

Deep learning architectures are inherently structured as unidirectional flows of information through sequential layers of abstraction. This characteristic naturally lends itself to visual representations, which are commonly found in DL research publications and educational materials. Recognizing that these architectures can be effectively visualized as sequences of interconnected nodes with specific properties, it is straightforward to explore the possibility that neural networks can be constructed and programmed in a visual manner. Supporting this assumption, in a quantitative survey among 113 software engineers and researchers from various backgrounds and experiences, Sankaran et al. reported that 72% of the respondents suggested that a **visual programming tool** would be useful to speed up the overall DL development process [42].

In traditional software development, visual programming is a paradigm that allows users to create programs by manipulating graphical elements rather than writing code textually. Researchers have made several attempts to introduce visual programming tools (VPTs) that enable an intuitive, "no-code" approach to designing deep learning models. DeepVisual [54], for example, is a PyCharm plugin enabling users to design neural networks through a drag-and-drop interface and supports bidirectional conversion between visual representations and Keras code. DARVIZ [42] introduced an abstract representation called Neural Language Definition Standard (NLDS) to enable interoperability across different deep learning frameworks. The authors of DL-IDE [47] expanded on these concepts and demonstrated that the development time and task accuracy using visual programming were better or comparable to coding. The authors also showed that DL-IDE caters to all users

irrespective of their prior expertise in deep learning, with a more marked advantage for novice users. These contributions, while valuable, exhibit two key limitations when considered in the context of current AI advancements: **scalability of model architectures** and **interactive debugging**. The scalability of model architectures is crucial as state-of-the-art networks can contain hundreds of layers with intricate substructures [6]. Existing tools lack hierarchical abstraction capabilities, making it difficult to build and visualize large, complex networks effectively. Interactive debugging during the design phase is another critical limitation of existing tools. While some offer basic checks, none provide instantaneous feedback on the validity and potential issues of the designed architecture. Implementing real-time validation would allow users to identify and resolve problems earlier in the development process, significantly enhancing the user experience and preventing errors during execution. In addition, while the authors of DL-IDE [47] demonstrated that time reduction when using VPTs compared to traditional coding was consistent across different levels of user expertise, they lacked a comprehensive qualitative assessment of how such tools are perceived by users with varying levels of expertise. Building upon prior contributions, we aim to address the identified gaps and introduce a system that aligns with current deep learning advancements. We introduce DeepFlow, a web-based VPT for Deep Learning Developers that shows the following contributions:

(1) To allow developers to design large and complex networks, we enable users to add, connect, and merge nodes into more sophisticated architectures. We introduce the concept of *supernodes*, which are composite nodes that can encapsulate multiple computational nodes or other supernodes, enhancing modularity and reusability in neural network design.
(2) We introduce *interactive debugging* for users to validate their architectures prior to training. This network verification feature identifies and highlights erroneous nodes in the User Interface, providing immediate visual feedback to users and guiding them in resolving errors before initiating the training process.
(3) We provide a *qualitative assessment* of user experiences with DeepFlow across different expertise levels. This evaluation explores the trade-offs between the flexibility of traditional coding and the accessibility of visual programming approaches for deep learning, with a particular focus on workflow integration for experienced users. By addressing this gap, we offer insights into how DeepFlow is perceived and utilized by a diverse user base, contributing to a more precise understanding of VPTs' role in the deep learning development ecosystem.

To align our discussion with existing contributions, DeepFlow includes the main functionalities of previous VPTs such as a wide range of PyTorch functions as nodes and export of created models in various formats (PyTorch, ONNX, JSON). We evaluated DeepFlow through a user study with 16 participants, including both Deep Learning novices and experienced practitioners. Participants were given a series of typical DL tasks to complete using DeepFlow. We assessed usability using the System Usability Scale (SUS) [5] and task difficulty using Single Ease Questions (SEQ) [43] to compare DeepFlow with coding. Results showed high usability scores and significantly lower perceived difficulty compared to coding approaches. Additionally, we gathered qualitative feedback through post-task interviews. Users found DeepFlow intuitive, praising its visual approach for rapid DL model prototyping and its value as an educational tool. Participants highlighted how the visual representation enhanced understanding of complex deep learning architectures. The analysis of user feedback also identified future directions for improvement, discussed at the end of the paper.

2 Related Work

Visual programming languages (VPLs) have emerged as powerful tools for democratizing access to complex computational processes, including machine learning (ML) and neural network design. These languages enable users to create programs by manipulating graphical elements rather than writing code textually, lowering the barrier to entry for those without extensive programming backgrounds. In ML applications, VPLs usually include nodes for data preprocessing and standard ML functions such as Principal Component Analysis (PCA), clustering, and classification algorithms. These tools focus on providing a range of components for traditional ML tasks, emphasizing data manipulation and classical algorithms. On the other hand, DL VPLs allow for the design of neural network architectures, enabling users to construct and visualize complex deep learning models.

The development of VPLs in ML applications has been driven by the need for user-friendly tools that can keep up with the increasing complexity of AI-based systems [28]. These tools mainly use graphical user interfaces (GUIs) to simplify the creation and manipulation of ML models [2, 33, 53]. A predominant trend in VPL-based ML tools is the use of flow-based programming languages. These languages represent computational processes as directed graphs, making complex data processing more intuitive through visual nodes and arcs [32]. Examples include Orange [10], a comprehensive data mining toolbox, and Goldenberry [16, 41], an extension of Orange for evolutionary algorithms. Block-based languages represent another significant category in VPL-based ML tools. These languages allow users to construct programs using drag-and-drop code blocks, reducing syntax errors and focusing on conceptual understanding [38]. Tools like Scratch and its implementations [35, 36, 38, 40, 45] have been widely used in educational settings to introduce ML concepts to novices. Other examples include ML Blocks [53] for creating TinyML models and Milo [39] for data science education. The application of VPL-based ML tools extends beyond educational settings, in industry, tools like PaddlePaddle [3] empower companies to train employees in both ML processes and business applications, aFlux [31] enables graphical Spark programming for IoT mashup tools. In healthcare, KNIME [51], RapidMiner [4], and Workflow Designer [24] have been used to develop ML-based systems for predicting hospital admissions, enhancing healthcare decisions, and designing EEG signal processing pipelines. Several VPL-based tools cater to specific ML tasks or domains. Visual Apriori [30] provides a visual interface for frequent itemset generation. Rapsai [12, 13] aims to accelerate ML prototyping of multimedia applications while incorporating features for model interpretability. Node-RED [29] has been used for IoT beehive monitoring with ML-based anomaly detection. OneLabeler [56] offers a flexible system for building data labeling tools.

Marcelle [15] allows for composing interactive ML workflows and interfaces. CO-ML [50] focus on collaborative ML model building, involving non-expert users in the process. General-purpose ML platforms with visual programming capabilities include WEKA [21], a comprehensive ML workbench; Yale [34] for rapid prototyping of complex data mining tasks; and Lemonade [11], a scalable Spark-based platform for data analytics. Tools like DeepGraph [22] focus on visualizing and understanding DL models, while Prompt Sapper [8] leverages large language models for building AI chains. The development of VPL-based tools for ML has seen innovations also in interaction modalities [50]. While drag-and-drop remains the primary mode of interaction for most tools [28], some researchers have explored alternative approaches. For instance, Mix & Match [23] integrates GUIs with tangible tokens, allowing users to manipulate physical objects to design ML-based systems. Similarly, Gest [19] employs sensors to engage children with ML concepts through gesture recognition. In the realm of deep learning (DL), which involves designing complex neural network architectures, several VPL-based tools have emerged to simplify the process. DeepVisual [55] provides a visual programming environment specifically for DL systems, enabling users to design neural networks through a drag-and-drop interface and supports bidirectional conversion between visual representations and Keras code. DL-IDE [48] offers a visual programming paradigm for abstract DL model development, introducing a drag-and-drop interface for constructing neural networks and implementing a framework-agnostic intermediate representation for exporting models to different DL libraries. Other tools like DeepBlocks [7], SMILE [26], and GraphicalAI [44] also aim to make DL development more accessible through visual interfaces. Recent tools like Talaria [20] focus on post-training optimization (e.g., model compression and hardware deployment analysis), while DeepFlow targets the earlier stages of model design, prototyping, and debugging. Unlike Talaria, which operates on pre-trained models, DeepFlow enables users to construct architectures interactively with hierarchical abstraction and real-time validation, addressing fundamentally different challenges in the ML development lifecycle. Existing VPL-based DL tools, however, exhibit several limitations. These limitations are motivated by the fact that since their development, new challenges have emerged in neural network design, most notably the need for model scalability to accommodate increasingly custom and complex architectures. Many tools present a limited and basic set of layer types and customization options, constraining users' ability to design complex architectures [52]. For instance, they primarily support basic layers like convolutional, pooling, and fully connected layers. The lack of hierarchical abstraction capabilities makes it difficult to build and visualize large, complex networks, which is crucial for building state-of-the-art architectures [6]. Another critical shortcoming is the lack of real-time validation and interactive debugging options. While some tools offer basic checks, they do not provide instantaneous feedback on the validity and potential issues of the designed architecture, leading to errors that only become apparent during code execution. Moreover, previous contributions lack thorough qualitative evaluations, limiting our understanding of how users with varying levels of expertise perceive and interact with these visual programming environments.

DeepFlow, the main contribution of the paper, addresses these challenges by enabling users to hierarchically construct neural networks, starting from low-level nodes and progressively building more complex architectures through adding, connecting, and merging layers, while maintaining visualizability and control over the entire hierarchy. It introduces interactive debugging options, allowing users to validate and debug their architectures prior to framework-agnostic export, providing instantaneous feedback. As such, DeepFlow is designed to be consistent with today's standards in deep learning development. Furthermore, we complement existing works by providing a qualitative assessment of the tool, offering insights into user experiences and perceptions across different expertise levels.

3 Design Principles

The development of DeepFlow was driven by a desire to address key challenges in the field of deep learning model creation. As deep learning becomes increasingly prevalent across various domains, it is essential to make the technology more accessible, provide tools that facilitate intuitive debugging, and ensure that models can scale to meet the demands of complex tasks. This section explores these design principles in depth, highlighting how they informed the creation of DeepFlow. These principles were developed through a combination of our analysis of existing literature on visual programming tools for deep learning [52] and our own experiences in addressing the limitations of current tools.

3.1 Balancing Accessibility with Advanced Functionality

DeepFlow is designed to provide accessibility for novices while offering functionality that benefits experienced practitioners. At its core, DeepFlow offers an intuitive drag-and-drop interface for constructing neural networks. Users create DL models by sequentially arranging nodes on a canvas, adding and connecting them to form the network architecture. This visual approach abstracts the complexities of the underlying code, allowing users to focus on the conceptual design of models rather than syntactical details [49]. The method not only simplifies the construction process but also improves network readability as it expands. DeepFlow offers a comprehensive library of pre-built components, representing common neural network layers and functions. Each component is accompanied by clear descriptions, making it easier for users to understand and utilize various elements of deep learning architectures. Each node in DeepFlow represents a specific PyTorch [1] function and includes configurable parameters. For beginners, the visual representation of neural networks helps to understand the structure and flow of data through different layers. More experienced users could benefit from the ability to rapidly prototype and experiment with different architectures without the overhead of writing extensive code.

3.2 Ensuring Scalability of Models

As the size of deep learning models needs to grow to reach higher performances, scalability becomes a critical concern [6]. Modern architectures may consist of hundreds of layers, making them difficult to manage and visualize using conventional tools. This complexity

[1] PyTorch (https://pytorch.org) is one of the most popular deep learning frameworks, widely used in both research and industry.

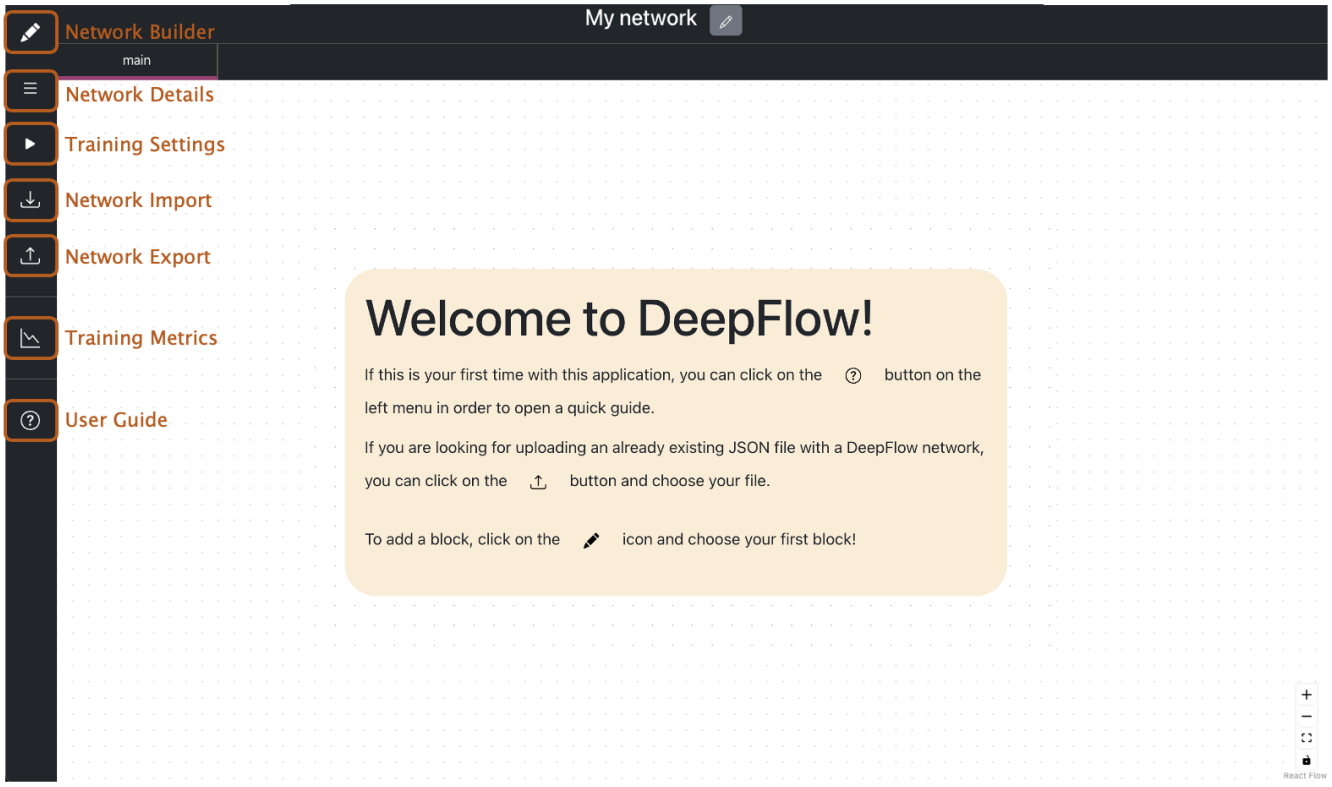


Figure 1: DeepFlow's main user interface, showcasing the canvas area and key functionalities with annotations.

can overwhelm users, hindering their ability to design, understand, and modify large-scale models effectively. Traditional interfaces often struggle to provide a clear overview of the model structure while still allowing for detailed manipulation of individual components [52]. Central to DeepFlow's approach to scalability is the concept of **supernodes**. Users can group multiple layers into these composite units, effectively encapsulating subsets of the network into manageable components. In the DeepFlow interface, a supernode appears as a single node on the canvas, but users can delve into it to reveal the individual nodes that constitute the supernode. This feature allows for a cleaner visual representation of the model, enabling users to navigate and edit complex architectures with ease. Supernodes can contain not only standard nodes but also other supernodes, allowing for multiple levels of abstraction. DeepFlow's user interface supports the creation and management of supernodes through multiple operations. Users can select multiple nodes and create a supernode, with the application automatically handling the encapsulation process. The interface also provides functionality to copy and paste supernodes, delete them, or modify their contents. By facilitating scalability in this way, DeepFlow ensures that users can design models that meet the demands of advanced applications without becoming overwhelmed by complexity. Furthermore, DeepFlow's scalability features significantly enhance its visualization capabilities. By allowing users to encapsulate complex sub-networks into supernodes, it provides an effective way to manage and visually comprehend model architectures. This hierarchical representation enables users to grasp the overall structure of advanced deep learning models while retaining the ability to examine or debug detailed components when needed.

3.3 Facilitating Intuitive Debugging

Debugging deep learning models poses significant challenges due to the abstract nature of neural networks and the delayed feedback typically received during training. Traditional development environments often lack real-time validation, leading to frustration and inefficiency as users encounter runtime errors that could have been prevented during the design phase. Additionally, these environments typically do not visualize the structure of neural networks, making it challenging for developers to quickly grasp the overall architecture and identify potential issues [42]. DeepFlow addresses this issue by integrating interactive debugging features directly into the model creation process. As users build their networks, DeepFlow provides immediate visual feedback on the validity of the architecture. The system employs an algorithm for continuous validation of the network structure. This algorithm processes the network graph to ensure compatibility and correctness at each step of model construction. It checks for potential errors such as mismatched input shapes, incompatible parameter settings, or incorrect connections between layers. When an error is detected, DeepFlow highlights the problematic node with a red outline and displays a detailed error message. This visual feedback allows users

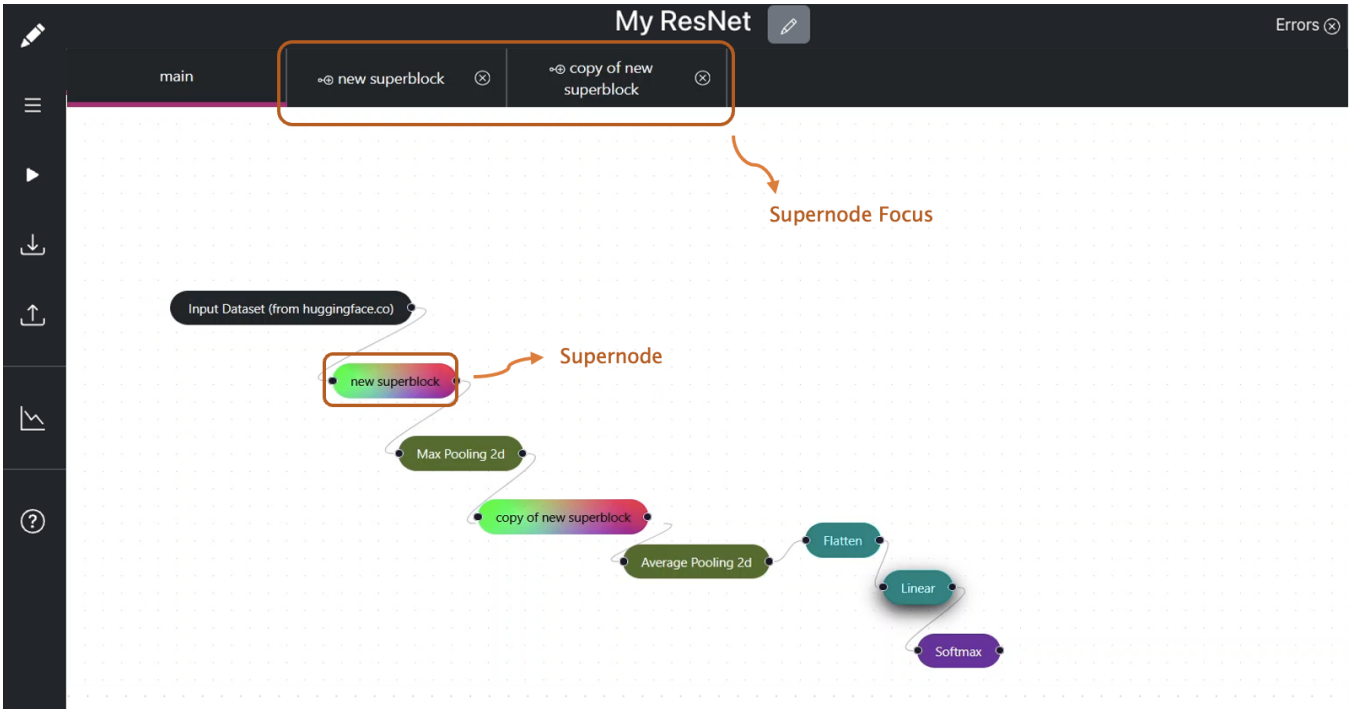


Figure 2: node-based model construction in DeepFlow, showing a partial ResNet [18] architecture.

to quickly identify and locate issues within their network structure. The error messages are designed to be informative, providing context about the nature of the problem and potential solutions. By catching errors early and providing detailed feedback we aim at reducing the trial-and-error typically associated with deep learning development.

4 DeepFlow: Implementation and Key Features

Building upon the design principles we implemented DeepFlow with features that align closely with these objectives and support effective user interaction patterns.

4.1 User Interface Overview

The user interface of DeepFlow is designed to provide an intuitive and accessible entry point for users. As shown in Figure 1, the main screen features a vertical menu bar on the left side, a layout that aligns with common interaction patterns, offering quick access to essential functions. In detail, the vertical menu includes options for *Network Builder*, which allows users to construct their neural network models; *Network Details*, for viewing and editing network specifications; *Training Settings*, where users can configure model training parameters; *Network Import/Export*, for saving or loading network designs; Training Metrics, to view performance results; and a *User Guide*, providing help and documentation. These functions align with DeepFlow's design objectives and features commonly found in VPTs for deep learning,

4.2 Node-Based Model Construction

DeepFlow employs a drag-and-drop interface for constructing neural networks, as illustrated in Figure 2. Users can add various types of layers and operations to their models, including input nodes (such as the *Input Dataset*), convolutional layers, pooling layers (like *Max Pooling 2d* and *Average Pooling 2d*), and other operations (such as *Flatten*, *Linear*, and *Softmax*). This visual method of model construction directly supports our design goal of accessibility by allowing users to build complex models without writing code. Connections between nodes are created by lines linking one node's output to another's input, as seen in the image. The system ensures compatibility between connected layers, supporting intuitive debugging by helping users identify and correct connection errors as they build their models. A key feature of DeepFlow is the concept of **supernodes**, represented by the "new superblock" and its copy in Figure 2. These supernodes allow users to encapsulate complex sub-networks, supporting scalability in model design. The supernode focus at the top of the screen provides quick access to created supernodes, enabling efficient navigation and management of large, complex architectures.

4.3 Supernode Creation and Scalability

To ensure the scalability of models and simplify the design of large networks, DeepFlow introduces the concept of *supernodes*, which encapsulate complex substructures within the network. As reported in Figure 3, users can select multiple nodes and create a supernode, grouping related components into a single, manageable entity. This feature supports scalability by allowing users to manage and navigate large networks more efficiently. Supernodes offer the

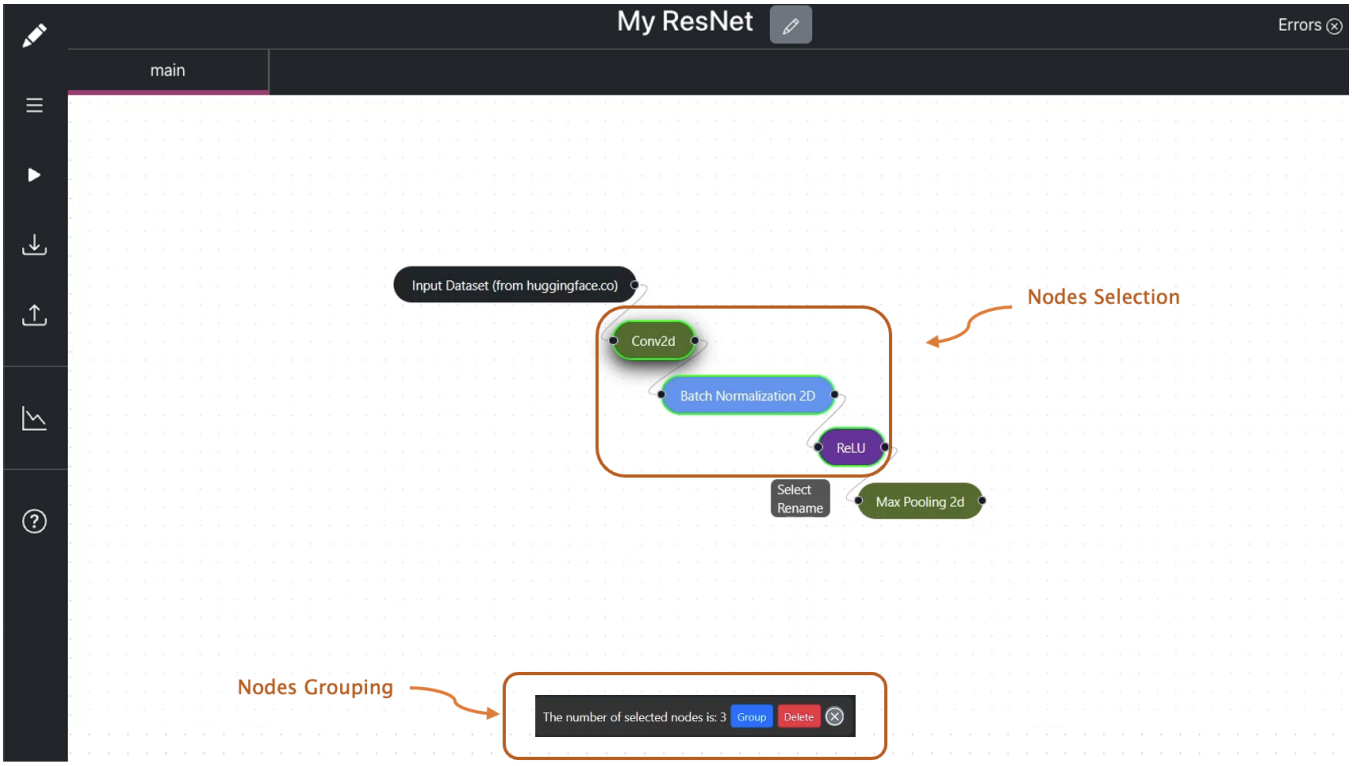


Figure 3: Supernode creation in DeepFlow, showing a residual node encapsulation.

advantages of modularity and abstraction. By encapsulating complex structures like residual nodes, users can focus on high-level architecture without losing access to lower-level details. These supernodes can be copied and pasted repeatedly throughout the network, facilitating the creation of repetitive structures common in deep learning architectures [6, 18]. This aligns with interaction patterns that favor modular design and reusability, enabling users to build complex models while maintaining clarity and control. To further enhance scalability, users can double-click on supernodes to view their content in detail, allowing for hierarchical exploration of model structures. This capability is complemented by zoom controls located at the bottom-right of the interface. These controls allow users to zoom in for detailed work on specific sections of the model, zoom out for an overview, and fit the entire network within a single screen view. Together, these functionalities provide a flexible way to navigate and edit large, complex models, supporting both high-level architecture design and detailed component configuration.

4.3.1 Example: Implementing Multi-Head Attention. To demonstrate DeepFlow's support for modern architectures, we outline the implementation of a multi-head attention mechanism using supernodes: (1) Construct a subgraph for one head by creating linear projections for queries, keys, and values (Q, K, V), followed by scaled dot-product computation (MatMul, Scale, and Softmax nodes). These components are grouped into a reusable *AttentionHead* supernode. (2) Replicate the *AttentionHead* supernode for multiple heads using DeepFlow's copy/paste functionality, adjusting input dimensions as needed. (3) Merge outputs from all heads using a Concat node, then apply a final linear projection layer. (4) Save the assembled multi-head attention supernode to the library for reuse in other architectures (e.g., transformers), as illustrated in Figure 4. This workflow highlights how hierarchical abstraction simplifies the design of complex, parallel structures while maintaining visual clarity.

4.4 Real-Time Debugging and Validation

DeepFlow incorporates real-time debugging features to help users identify and resolve issues quickly. The system employs the algorithm, shown in Algorithm 1, for continuous validation of the network structure as users build or modify their models.

The algorithm processes each node in a topologically sorted order, ensuring that all dependencies are satisfied before a node is evaluated. It maintains a data structure that keeps track of the current state of each terminal in the network. As each node is processed, the algorithm checks for potential errors such as mismatched input shapes, incompatible parameter settings, or incorrect connections between layers.

As shown in Figure 4, when an error is detected, the problematic node is highlighted with a red outline, and a detailed error message is displayed. This visual feedback allows users to quickly identify and locate issues within their network structure. The real-time nature of this validation process creates an immediate feedback loop, aiming to reduce the time spent on troubleshooting. Users can learn

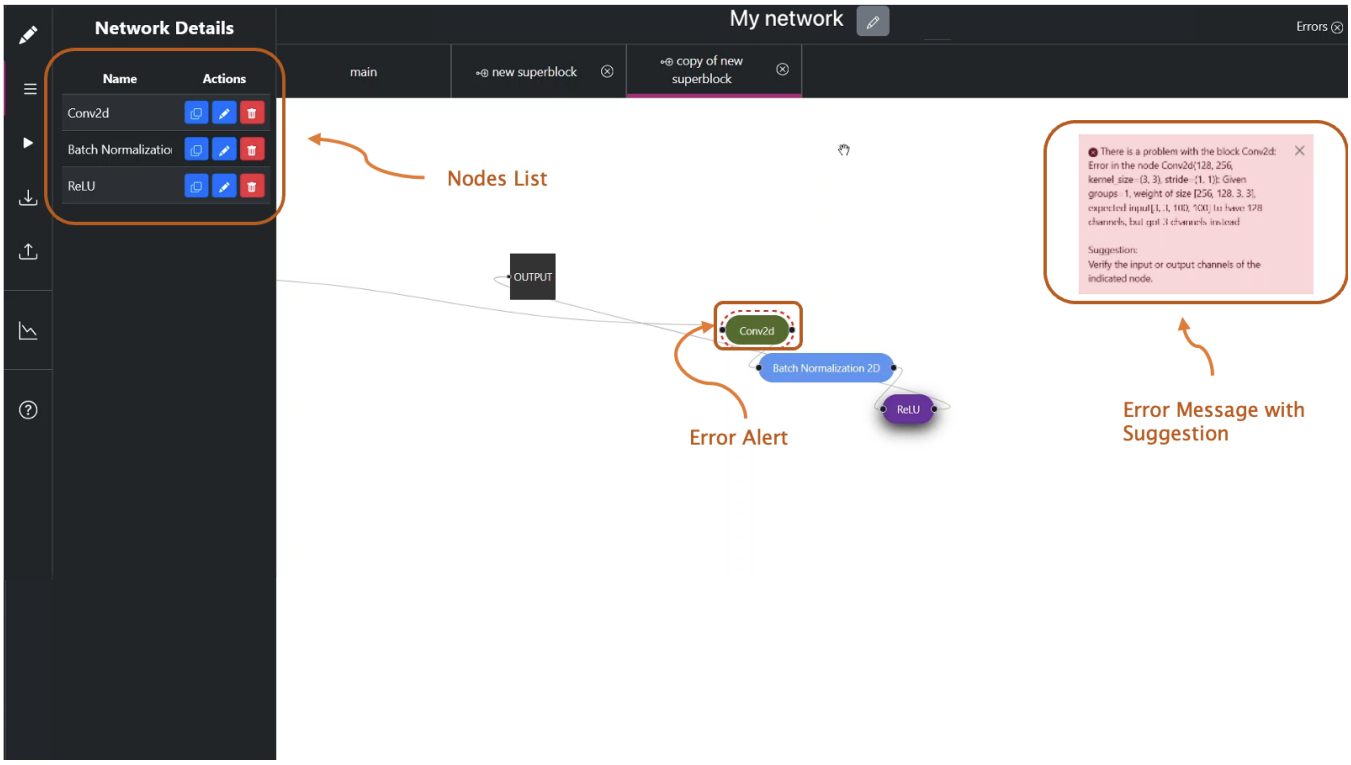


Figure 4: Real-time debugging in DeepFlow, highlighting an error in the Conv2d node.

Algorithm 1 Real-Time Network Validation

```
procedure ValidateNetwork(network)
    sortedNodes ← TopologicalSort(network)
    data ← InitializeInputData(network)
    for each node in sortedNodes do
        inputData ← GatherInputData(node, data)
        outputData ← ProcessNode(node, inputData)
        UpdateDataState(data, node, outputData)
        HighlightError(node)
    end for
    return true
end procedure
```

from mistakes in real-time and adjust their designs accordingly, ensuring a smoother development experience and encouraging design exploration.

By integrating this debugging mechanism into the model construction process, DeepFlow implements the design principle of intuitive debugging. This approach is particularly beneficial for novice users, helping in error detection but also aiding in understanding the relationships between different layers to learn the fundamentals of neural network architecture. For experienced practitioners, it facilitates iterating fast on complex model designs.

4.5 Additional Features

To ensure a comprehensive tool and enable fair comparison with existing solutions, DeepFlow implements several additional features that are common in VPTs for deep learning:

4.5.1 Training Configuration and Results Visualization. Although our tool primarily focuses on the design of neural network architectures, for fair comparison with existing tools, we implemented additional features related to model training and evaluation. Once a model is constructed and validated, users can configure training parameters and initiate the training process. DeepFlow provides visualization of results, including training and validation loss curves, accuracy metrics over epochs, confusion matrices for classification tasks, and layer-wise activation visualizations.

4.5.2 Export and Interoperability. DeepFlow supports exporting models in various formats, such as PyTorch (.pth) for direct use in PyTorch environments, ONNX format for interoperability with other frameworks, and JSON representation for easy sharing and version control. This feature ensures that models created in DeepFlow can be seamlessly integrated into existing workflows or deployed in production environments. By facilitating interoperability, DeepFlow enhances scalability and supports users as they advance to more complex projects or integrate with other tools. This aligns with interaction patterns that value flexibility and adaptability, allowing users to transition smoothly between different stages of development.

5 User Study

Building on findings from [49], which demonstrated that visual programming reduces development time and improves task completion rates compared to coding, this study focuses on novel capabilities like hierarchical abstraction and debugging. Our results extend these prior insights by exploring how these features are perceived across expertise levels. To evaluate the effectiveness of DeepFlow in facilitating deep learning model development, particularly through its scalability and debugging features, we conducted a user study involving 16 deep learning practitioners with varying levels of expertise. This choice of diverse expertise levels is motivated by the multiple advantages that visual programming languages have demonstrated, both in enhancing rapid prototyping for expert users and in improving accessibility for novices in the field of deep learning [49]. Our methodology combines quantitative measurements with qualitative insights to provide an understanding of both the efficacy of the proposed tool and users' experiences.

5.1 Participants

We recruited participants using a combination of snowball and convenience sampling, focusing on individuals with an interest or experience in machine learning and deep learning. Prospective participants completed an initial questionnaire covering:

- *Demographic and background information:* age, gender, educational background, current occupation, and experience with machine learning.
- *DL proficiency:* self-rated confidence in machine learning concepts (novice, beginner, intermediate, advanced, expert).
- *Coding frequency:* how often they engage in DL programming (less than once a month to more than once a week).

Participants qualified for the study if they met the following criteria: a) currently employed or studying in a field related to computer science, engineering, or data science; b) proficiency in English; c) some level of familiarity with machine learning concepts; and d) experience in coding or implementing DL models. We aimed for a diverse sample in terms of expertise levels, backgrounds, and current roles to ensure a range of perspectives. Our final sample consisted of 16 participants (13 males and 3 females) with ages ranging from 19 to 30 years old. The majority (12) were in the 22-26 age range, with 2 participants aged 19-21, and 2 aged 27-30. The sample represented a variety of educational backgrounds, with 5 participants pursuing bachelor's degrees, 2 pursuing master's degrees, 7 pursuing or holding doctoral degrees, and 2 employed as software engineers. Participants had varying levels of DL experience and expertise. In terms of DL confidence, 5 participants rated themselves as expert, 4 as advanced, 5 as intermediate, 1 as beginner, and 1 as novice. The sample included individuals from diverse backgrounds, including biomedical engineering (4), computer engineering (8), electronic engineering (1), and others (3). All participants reported some level of experience with machine learning concepts and coding. The frequency of DL programming varied widely among participants, from less than once a month (5 participants) to more than once a week (3 participants). This diversity in expertise levels, from novice students to experienced researchers and professionals, provides an optimal context for evaluating the effectiveness and user interaction with DeepFlow across a wide spectrum of deep learning proficiency. Table 1 provides an overview of the sample, including their occupations, DL confidence levels, and coding frequencies.

5.2 Procedure

We conducted the study in individual sessions, each lasting approximately one hour. The study was structured into three main phases:

Introduction and consent. Participants were briefed on the study's purpose and procedures, and provided informed consent. They were reminded that the study aimed to evaluate the tools, not their personal performance.

Task completion. Participants were tasked with constructing a ResNet [18] architecture using both traditional coding methods and DeepFlow. ResNet was selected for its historical significance in enabling scalability through the introduction of residual connections, a concept central to modern deep learning architectures. Its structure, composed of repeating residual blocks, serves as an ideal test case for DeepFlow's superblock feature, allowing us to assess the tool's effectiveness in managing complex, scalable designs. Moreover, ResNet's prevalence in both academic and industrial settings ensures that it provides a familiar reference point for participants across all levels of expertise. To mitigate potential learning effects, the order of these tasks was counterbalanced among participants.

Post-task evaluation. After completing each task, participants answered a Single Ease Question (SEQ) [43], rating the difficulty of the task on a 7-point Likert scale. Following all tasks, participants filled out the System Usability Scale (SUS) [5] questionnaire for DeepFlow. They also participated in a semi-structured interview to gather more in-depth feedback on their experience, discussing potential impacts of the tool on their deep learning workflow.

5.3 Measurements

We collected both quantitative and qualitative data to assess the effectiveness of DeepFlow compared to traditional coding:

Success rate. We recorded whether participants successfully completed the ResNet model design task using traditional coding and each subtask using DeepFlow, providing insights into the effectiveness of each method.

Error count. We tracked the number of critical and non-critical errors made during task completion for both the traditional coding approach and DeepFlow subtasks.

Single Ease Question (SEQ). After each task, participants rated the perceived difficulty on a 7-point Likert scale, providing immediate feedback on task complexity.

System Usability Scale (SUS). The SUS scores provided a quantitative measure of perceived usability for DeepFlow.

Qualitative feedback. The think-aloud protocol during task completion and the semi-structured interviews explored participants' experiences, preferences, perceived strengths and weaknesses of each method, and suggestions for improvement.

This combination of measurements allowed us to assess both the task performance and subjective experiences of participants

Table 1: An overview of the 16 participants in our user study, including their demographic information, occupation, and self-reported experience with DL and coding frequency.

ID (Gender, Age)	Occupation	DL Confidence	DL Coding Frequency
P1 (M, 23)	M.S. Student in Biomedical Engineering	●●●○○	●●●○○
P2 (M, 25)	Student in Biomedical Engineering	●●●●●	●●○○○
P3 (M, 24)	Student in Biomedical Engineering	●●●●●	●●●●○
P4 (F, 25)	Student in Computer Engineering	●●●●●	●●●●●
P5 (M, 22)	M.S. Student in Computer Science	●●●●○	●●●●○
P6 (M, 25)	Researcher in Cinema Engineering	●○○○○	●○○○○
P7 (M, 24)	Ph.D. Student in Computer Engineering	●●●○○	●○○○○
P8 (M, 24)	Student in Biomedical Engineering	●●●●○	●●●○○
P9 (F, 30)	Research Assistant in Bioengineering	●●●○○	●●●○○
P10 (F, 27)	Post-Doc Researcher in DL	●●●●●	●●●●●
P11 (M, 19)	Student in Computer Engineering	●●●○○	●●○○○
P12 (M, 26)	Ph.D. Student in Computer Engineering	●●○○○	●○○○○
P13 (M, 28)	Ph.D. Student in Computer Engineering	●●●●●	●●○○○
P14 (M, 25)	Software Engineer	●●●●●	●○○○○
P15 (M, 25)	Ph.D. Student in Computer Engineering	●●●●●	●●●●●
P16 (M, 23)	Software Engineer	●●●○○	●●●●○

●○○○○ = Novice, ●●○○○ = Beginner, ●●●○○ = Intermediate, ●●●●○ = Advanced, ●●●●● = Expert

●○○○○ = Less than once a month, ●●○○○ = Once a month, ●●●○○ = Less than once a week, ●●●●○ = Once a week, ●●●●● = More than once a week

across different expertise levels when using DeepFlow compared to traditional coding methods. While direct time comparisons between the methods were not feasible due to the different task structures, our focus on qualitative feedback and task-specific metrics provided valuable insights into the relative strengths and potential areas for improvement in each approach.

5.4 Data Analysis

We employed a mixed-methods approach to analyze our data, combining quantitative statistical analyses with qualitative examination of participant feedback.

Quantitative analysis: We used descriptive statistics to summarize the success rates for both the traditional coding task and DeepFlow subtasks. The SEQ responses for DeepFlow subtasks were analyzed using means and standard deviations to assess perceived task difficulty. A paired t-test was conducted to compare the difficulty of tasks performed using coding versus DeepFlow. System Usability Scale (SUS) score was calculated and interpreted according to standard SUS guidelines.

Qualitative analysis: For the interview data, we focused on extracting relevant quotes and insights that reflected participants' experiences. We reviewed the interview transcripts, identifying statements that provided rich descriptions of participants' thoughts, preferences, and suggestions regarding their use of DeepFlow and how it compared to their usual coding practices. These quotes were then organized thematically to complement and provide context for our quantitative findings, offering a deeper understanding of participants' interactions with DeepFlow.

6 Results

This section presents both quantitative and qualitative results from our user study.

6.1 Quantitative Results

The System Usability Scale (SUS) questionnaire results, as shown in Figure 5, indicate a generally positive user experience with DeepFlow. The overall SUS score of 79.53 falls into the "good" range of usability, suggesting that users found the system largely intuitive and effective. Key observations from the SUS results reveal high user confidence and a desire for frequent system use. Participants generally perceived the system as easy to use with well-integrated functions. However, there was some indication that technical support might be needed for certain users. Opinions on the learning curve were mixed, with some users feeling they needed to acquire significant knowledge before effectively using the system. This analysis highlights DeepFlow's strengths in user-friendliness and functionality, while also identifying areas for potential improvement in user onboarding and support.

Figure 6 presents the results of the Single Ease Question (SEQ) comparing the perceived difficulty of completing tasks using traditional coding versus DeepFlow. The average difficulty score for coding was 3.31 out of 5 (SD = 0.89), while for DeepFlow it was 2.12 (SD = 0.76). This significant difference ($p < 0.05$) indicates that participants found tasks considerably easier to complete using DeepFlow compared to traditional coding methods. The lower score for DeepFlow suggests that its visual programming approach and

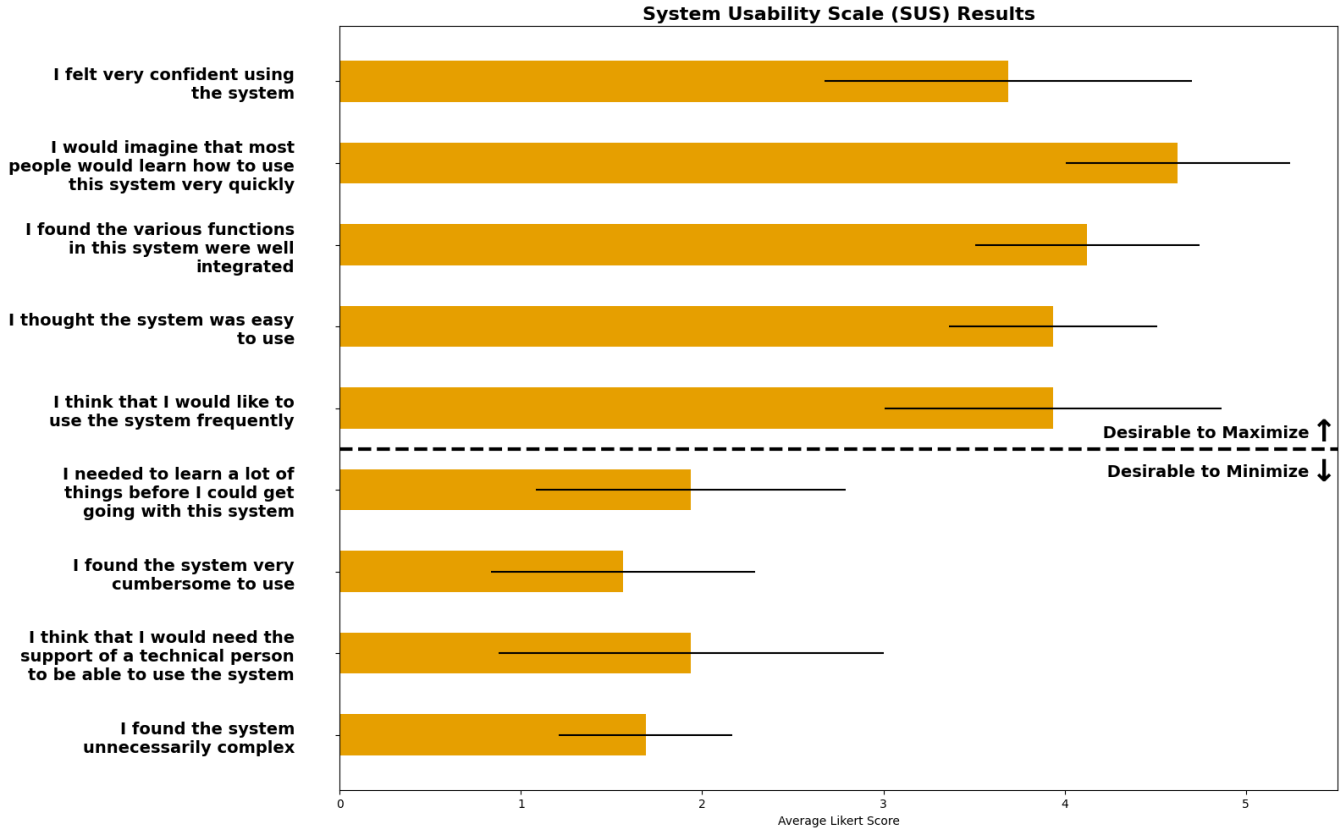


Figure 5: System Usability Scale (SUS) questionnaire results for DeepFlow.

intuitive interface effectively reduced the perceived complexity of creating deep learning models.

Figure 7 provides a more detailed breakdown of task difficulty across different user groups. Notably, while novice users reported significantly higher difficulty levels in coding tasks compared to experts, the difficulty levels for DeepFlow tasks were more comparable between novices and experts. For this analysis, we split the participants into two groups: novices (self-reported DL confidence ≤ 3) and experts (self-reported DL confidence > 3). This observation underscores DeepFlow's potential to make deep learning model creation more accessible by reducing the perceived complexity difference between experienced and novice users. The visual programming approach appears to effectively lower the barrier of entry for beginners while still providing a useful tool for more experienced practitioners.

6.2 Qualitative Results

To evaluate the usability and effectiveness of DeepFlow, we conducted an analysis of participant responses (P1–P16) to semi-structured questions.

6.2.1 Ease of Use and Intuitiveness. A predominant theme among participants was the ease of use and intuitiveness offered by DeepFlow's. Many participants found that the graphical user interface simplified the process of building deep learning models, making it more accessible, especially for those less familiar with coding.

Participant P1 (novice) highlighted this by stating, *"DeepFlow simplifies the process of creating a deep learning model by providing a graphical user interface that enables me to intuitively translate what I have in my mind into a model."* This sentiment was echoed by P2 (expert), who mentioned, *"It's much easier and faster than writing code."*

Participant P4 (expert) found the tool intuitive and appreciated the speed of development: *"It was quicker and more intuitive. The possibility to visualize the nodes and connect them helped me to easily understand the flow of the model."* Similarly, P6 (novice) noted, *"You can build a model faster and set the parameters easily."*

6.2.2 Enhanced Visualization and Understanding. The visual nature of DeepFlow was recognized as a significant advantage for understanding and conceptualizing deep learning architectures. Participants appreciated the ability to see the model's structure and flow, which aided in comprehension and communication.

P3 (expert) observed, *"DeepFlow made it easier to visualize the steps of creating a model with each layer pass compared to raw code."* P4 (expert) also emphasized the benefit of visualization: *"The possibility to visualize the nodes and connect them helped me to easily understand the flow of the model."*

P12 (novice), who lacked prior deep learning experience, found the approach advantageous: *"Not having experience in creating deep learning models, I can say that I like the app's approach. It seems useful*

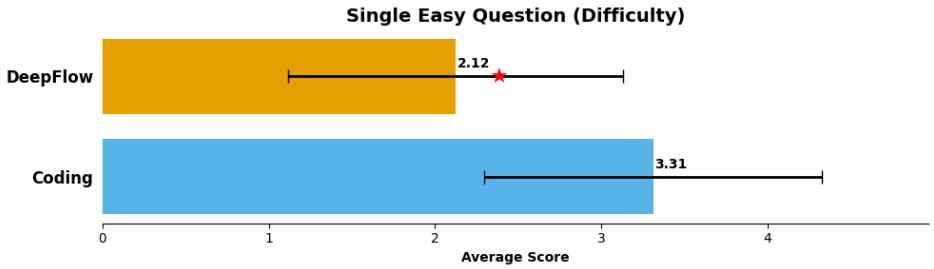


Figure 6: Comparison of average task difficulty scores between coding and using DeepFlow.

because it replaces repetitive programming tasks that are convenient to perform 'on the fly' graphically."

6.2.3 Prompt Feedback and Error Prevention. Participants valued the immediate feedback provided by DeepFlow, which helped prevent errors during model creation. The tool's ability to check for correctness in layer connections and parameter configurations was seen as a beneficial feature. P1 (novice) mentioned, *"Having a clear block-wise list of parameters that I have to define allows me not to miss any. In the end, prompt feedback on the correctness of the links between one layer and the other helped me a lot in avoiding modeling errors."* P3 (expert) also appreciated the system's checks: *"The system was useful in providing checks to make sure the model didn't have any mismatched parameters between nodes."*

6.2.4 Impact on Workflow and Productivity. Participants had mixed opinions on how DeepFlow would impact their workflow if used regularly. Many believed it could enhance efficiency and streamline the development process, while others felt it might not integrate well with their existing practices.

P1 (novice) anticipated a positive impact: *"I think that using DeepFlow regularly would improve my workflow because I could create a new model by visualizing a previous one I created for a similar task and graphically modify the nodes with direct feedback."* P4 (expert) agreed, stating, *"It would make the process of creating a deep learning model more efficient, and it would also allow for experimenting with various hyperparameters and different configurations of the model within a single environment."*

Conversely, P5 (expert) felt that DeepFlow might not fit into their workflow: *"I prefer to write code. I think of it as a learning tool and not a programming tool."* P15 (expert) shared this perspective, noting that it would likely slow them down: *"It would likely slow me down as it is not the same as programming."*.

6.2.5 Comparison with Traditional Coding. When comparing the visual programming approach of DeepFlow to traditional coding, participants highlighted both advantages and disadvantages. On the positive side, the visual approach was seen as more accessible and easier for beginners. P1 (novice) stated, *"It makes it much easier and faster."* P12 (novice) added, *"It is definitely better for those who would like to have a visualization of the whole model."*

However, some experienced programmers preferred traditional coding methods, citing familiarity and the efficiency of code-centric tools. P8 (expert) commented, *"Usually I just program my DL models. I am just more comfortable doing everything by code at this point; a visual system does not add much value to my process, if anything it slowed me down, because it does not have stuff like code completion or the shortcuts that I am used to using."* P5 (expert) commented, *"I think it fits perfectly to teach the basics of deep learning but not as a tool that programmers would use every day."*. P5 (expert) also felt that changing parameters was easier when directly manipulating code: *"I feel that the biggest drawback is changing the parameters for each layer. I think looking directly at the code makes it easier to change parameters."*. P13 (expert) emphasized the efficiency of coding: *"In my view, traditional coding is hard to beat, and things that rely on using a mouse pointer usually slow things down."*.

6.2.6 Potential Use Cases. Many participants identified education and beginner projects as ideal scenarios for using DeepFlow. The tool's visual and intuitive nature makes it well-suited for teaching deep learning concepts and lowering the barrier to entry for newcomers.

P3 (expert) noted, *"DeepFlow would be best used for beginner projects for those just getting into deep learning, as it would help users visualize and understand how deep learning works."* P5 (expert) echoed this sentiment: *"I think that it would be good for learning and teaching."*

P11 (novice) emphasized the tool's accessibility: *"The visual programming allows access to DNNs even to people with shortcomings or difficulties in programming. Think of a person approaching this sector with little or no prior knowledge."*. Participants believed that tools like DeepFlow could significantly impact the field by making deep learning more accessible and shifting the focus from coding to model design. P1 (novice) suggested that such tools *"will enable programmers in different areas, not especially experts in DL, to include DL in their processing pipelines."*

P4 (expert) envisioned improved efficiency: *"It could help developers to easily experiment with changes in the models in very few seconds, making efficient the model creation and improvement."* P7 (novice) reflected on the broader implications: *"I think that node-based programming is a super interesting concept because, if well made, systems that allow its use not only simplify the work of those who work in a certain field every day (e.g., deep learning) but also allow more people to approach that field, perhaps people intimidated by some aspects of programming."*

6.2.7 Challenges and Limitations. Participants provided valuable suggestions for enhancing DeepFlow to better meet their needs. Key recommendations included improving the intuitiveness of advanced features, enhancing the user interface, and expanding functionality.

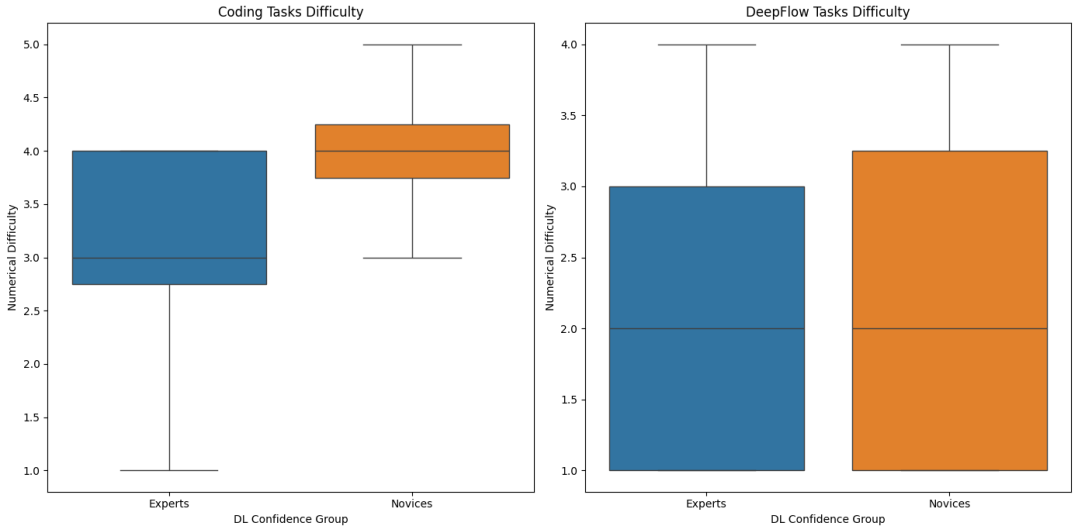


Figure 7: Comparison of task difficulty between experts and novices for coding and DeepFlow tasks.

P6 (novice) suggested integrating common architectures: *"It could be improved by adding common architectures already available in PyTorch, for example, and allowing them to be imported without the need of constructing them from scratch."*. P4 (expert) proposed expanding the tool's capabilities: *"DeepFlow could be enhanced with the introduction of new nodes and add links to the documentation of the framework (PyTorch, TensorFlow) of each block. Another improvement could be the possibility of downloading the whole code, including the training procedure and not only the model, could be very helpful."*. Participants identified several usability issues in DeepFlow, particularly with the implementation of supernodes and certain interface elements. Additionally, participants reported difficulties with node connections, a lack of expected keyboard shortcuts, and disruptive error messages. These findings highlight areas for improvement in the user interface design.

7 Discussion

Our study of DeepFlow reveals several key insights into the potential and limitations of visual programming tools for deep learning. The overall System Usability Scale score indicates that participants generally found DeepFlow to be user-friendly and effective. The comparison of task difficulty scores between traditional coding and DeepFlow demonstrates a significant reduction in perceived difficulty when using the visual interface. Notably, the reduction in perceived difficulty was consistent across different levels of user expertise, suggesting that DeepFlow successfully bridges some of the gaps between novice and experienced users, as also shown in previous studies [48].

The qualitative analysis highlighted several insights into user perceptions of the tool. Experienced users, while appreciating DeepFlow's intuitive design, expressed concerns about potential efficiency losses in their established workflows. This preference was not solely based on habit, but on perceived limitations in DeepFlow's ability to handle edge cases and highly-customized implementations. Addressing these concerns could be a valuable research direction to improve DeepFlow for experienced users, while not impacting its usage by novices. This finding suggests that visual programming tools in deep learning may not replace traditional coding entirely, but being most beneficial for experienced users when integrated with conventional coding approaches, rather than substituting them. A hybrid approach could potentially satisfy both the need for intuitive design and the demand for granular control. While our study emphasizes qualitative feedback for new features, prior work [49] established quantitative advantages of visual programming over coding. Future work will include comparative benchmarks (e.g., time-to-prototype, error rates) across expertise levels to further validate these findings.

This observation hints at a broader discussion in human-computer interaction on how we can design tools that adapt to increasing expertise without becoming cumbersome [14, 46].

Results demonstrated DeepFlow's intuitive interface and its ability to visualize complex model architectures. The visual nature of DeepFlow encouraged a more holistic, top-down approach to architecture planning, contrasting with the often bottom-up, layer-by-layer construction typical in code-based development. Such a shift in perspective could potentially lead to more innovative and efficient model designs, as users can more easily conceptualize and experiment with high-level structural changes. In terms of debugging, the visual representation made architectural flaws more immediately apparent than in traditional code. Several experienced users noted enhanced error detection capabilities, suggesting that visual programming tools are a valuable complementary tool for model debugging and optimization. In addition, the rapid prototyping capabilities offered by DeepFlow could accelerate the early stages of model development, even for experienced practitioners. By allowing quick visualization and manipulation of complex architectures, DeepFlow enables a more iterative and exploratory approach to model design. This aligns with the concept of "exploratory programming" [25], where rapid prototyping leads to more innovative solutions.

Finally, the tool's ability to make abstract deep learning concepts tangible through visual representation can bridge the gap between theoretical understanding and practical implementation. This is particularly relevant in educational settings, where the disconnect between mathematical formulations and their software implementations often poses a significant hurdle [27]. DeepFlow's visual approach might serve as an intermediate step, helping students translate theoretical knowledge into practical model architectures more intuitively.

It is crucial to note that there is a risk that users, especially novices, might rely too heavily on the visual interface without fully understanding the underlying principles. Future developments should consider how to balance ease of use with opportunities for improving the understanding of underlying mechanisms, perhaps by incorporating interactive tutorials or 'explainable artificial intelligence' features that elucidate the reasoning behind model structures [17].

7.1 Evaluation Scope and Limitations

Our evaluation focused on qualitative insights into DeepFlow's novel capabilities (e.g., supernodes, interactive debugging) and their perceived utility across expertise levels. This approach aligns with our goal of understanding how visual abstraction and real-time validation impact user workflows, particularly for complex architectures. While prior work [49] established quantitative advantages of visual programming, our study intentionally prioritized qualitative metrics to uncover nuanced user experiences, such as how hierarchical abstraction aids architectural scalability or how debugging feedback reduces cognitive load. However, this scope introduces limitations. First, our participant pool (16 users) and task design (ResNet implementation) limit generalizability to more diverse architectures (e.g., transformers, graph networks) or specialized domains. Second, while we measured perceived difficulty via SEQ and usability via SUS, we did not directly compare time-to-prototype or error rates between DeepFlow and traditional coding—metrics critical for quantifying productivity gains. Third, the study's controlled environment may not fully reflect real-world scenarios where users iteratively refine models over days or weeks.

8 Conclusion and Future Work

This paper introduced DeepFlow, a flow-based visual programming tool designed to enhance deep learning development by addressing challenges in scalability and debugging. Key contributions of DeepFlow include the introduction of supernodes for hierarchical abstraction, enabling efficient management of complex neural network architectures; interactive debugging capabilities providing real-time validation of network designs; and a qualitative analysis revealing insights into the tool's effectiveness and limitations across different user groups. Our evaluation, involving participants with varying levels of expertise, demonstrated DeepFlow's effectiveness in reducing the perceived difficulty of deep learning model creation across user groups.

We identified the need to investigate whether and how DeepFlow can be extended to offer the same level of flexibility as traditional coding environments. This includes exploring advanced customization features that allow more granular control over model components and parameters for expert users. Adapting DeepFlow to support emerging paradigms in machine learning, such as federated learning or reinforcement learning, could be another promising avenue for meeting expert users' requirements. Future research will conduct large-scale performance comparisons between DeepFlow and traditional coding, measuring metrics like development speed, model complexity, and debugging efficiency across user groups. Additionally, we will focus on integrating visual programming with traditional coding, potentially through a hybrid interface that combines visual elements with textual coding. This approach could bridge the gap between the intuitive design of visual programming tools and the detailed control required by experienced practitioners. In the context of rapid prototyping, DeepFlow's potential extends beyond individual use to collaborative scenarios. The visual nature of the tool could facilitate communication between team members with varying levels of technical expertise, fostering a more inclusive and efficient collaborative design process.

In conclusion, DeepFlow builds upon the demonstrated potential of visual programming tools in making deep learning more accessible and efficient [49]. By addressing common pain points such as model scalability and debugging, as well as assessing feedback from users with different expertises, this work takes steps toward simplifying the complexities inherent in neural network development and reassessing user needs across different expertise levels. As deep learning continues to evolve and permeate various domains, tools like DeepFlow can serve as catalysts for innovation by streamlining development processes, lowering barriers to entry, and fostering a more inclusive community of practitioners. This research contributes to the ongoing effort to democratize deep learning and suggests potential directions for future work in visual programming interfaces for artificial intelligence development.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] R Anil, R Danymol, H Gawande, and R Gandhiraj. 2014. Machine learning plug-ins for GNU Radio Companion. In *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*. IEEE, 1–5.

[3] Ran Bi, Tong Xu, Ming Xu, and Enhong Chen. 2022. PaddlePaddle: A Production-Oriented Deep Learning Platform Facilitating the Competency of Enterprises. In *2022 IEEE 24th International Conference on High Performance Computing & Communications; 20th International Conference on Smart City; 8th International Conference on Data Science & Systems (HPCC/SmartCity/DSS)*. IEEE, 92–99.

[4] Mohamed Bjaoui, Hela Sakly, Moemen Said, Naoufel Kraiem, and Mohamed Salim Bouhlel. 2020. Depth insight for data scientist with RapidMiner «an innovative tool for AI and Big Data towards medical applications». In *Proceedings of the 2nd International Conference on Digital Tools & Uses Congress*. 1–6.

[5] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan,

and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[7] Tobia Calò and Luigi De Russis. 2023. Towards a Visual Programming Tool to Create Deep Learning Models. In *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 38–44.

[8] Yuyang Cheng, Jialun Chen, Qingyun Huang, Zhenchang Xing, Qingkai Xu, and Qinghua Lu. 2023. Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains. In *ACM Transactions on Software Engineering and Methodology*.

[9] François Chollet et al. 2018. Keras: The python deep learning library. *Astrophysics source code library* (2018), ascl–1806.

[10] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, et al. 2013. Orange: data mining toolbox in Python. *The Journal of Machine Learning Research* 14, 1 (2013), 2349–2353.

[11] Walter dos Santos, Lucas FM Carvalho, Gustavo de P Avelar, Alberto Silva, Leticia M Ponce, Dorgival Guedes, and Wagner Meira. 2017. Lemonade: A scalable and efficient spark-based platform for data analytics. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 745–748.

[12] Ruofei Du, Naixi Li, Jiatong Jin, Michelle Carney, Sam Miles, Michael Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingwei Li, et al. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–23.

[13] Ruofei Du, Naixi Li, Jiatong Jin, Michelle Carney, Xiuxiu Yuan, Rahul Iyengar, Peng Yu, Adarsh Kowdle, and Alex Olwal. 2023. Experiencing Rapid Prototyping of Machine Learning Based Multimedia Applications in Rapsai. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–4.

[14] Leah Findlater and Joanna McGrenere. 2009. Design and evaluation of a command recommendation system for software applications. In *ACM Transactions on Computer-Human Interaction (TOCHI)*, Vol. 16. ACM, 1–36.

[15] Jules Françoise, Baptiste Caramiaux, and Téo Sanchez. 2021. Marcelle: composing interactive machine learning workflows and interfaces. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 39–53.

[16] Leandro Perez Garzón-Rodriguez, Hernán A Diosa, and Sergio Rojas-Galeano. 2015. Deconstructing GAs into visual software components. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1125–1132.

[17] David Gunning and David Aha. 2019. DARPA's explainable artificial intelligence (XAI) program. *AI Magazine* 40, 2 (2019), 44–58.

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

[19] Tom Hitron, Yoav Orlev, Iddo Wald, Ariel Shamir, Hadas Erel, and Oren Zuckerman. 2019. Can children understand machine learning concepts? the effect of uncovering black boxes. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–11.

[20] Fred Hohman, Chaoqun Wang, Jinmook Lee, Jochen Görtler, Dominik Moritz, Jeffrey P Bigham, Zhile Ren, Cecile Foret, Qi Shan, and Xiaoyi Zhang. 2024. Talaria: Interactively Optimizing Machine Learning Models for Efficient Inference. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 648, 19 pages. https://doi.org/10.1145/3613904.3642628

[21] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. 1994. Weka: A machine learning workbench. In *Proceedings of ANZIIS'94-Australian New Zealnd Intelligent Information Systems Conference*. IEEE, 357–361.

[22] Quan Hu, Lei Ma, and Jianjun Zhao. 2018. DeepGraph: A pycharm tool for visualizing and understanding deep learning models. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 628–632.

[23] Anneli Jansen and Simone Colombo. 2023. Mix & Match Machine Learning: An Ideation Toolkit to Design Machine Learning-Enabled Solutions. In *Proceedings of the Seventeenth International Conference on Tangible, Embedded, and Embodied Interaction*. 1–18.

[24] Petr Ježek and Lukáš Vařeka. 2019. Workflow Designer-a web application for visually designing EEG signal processing pipelines. In *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 368–373.

[25] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2017), 25–29.

[26] Ivan Khodnenko, Sergey V Ivanov, and Anna Lantseva. 2020. A lightweight visual programming tool for machine learning and data manipulation. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 981–985.

[27] Byung-Hoon Kim, Jungkook Kim, Hyeoksoo Kim, Youngrok Kim, and Junmo Kim. 2018. Teaching artificial intelligence and machine learning in a visual and interactive manner. *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (2018), 836–840.

[28] Mohammad Amin Kuhail, Saad Farooq, Rawad Hammad, and Mohammed Bahja. 2021. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access* 9 (2021), 14181–14202.

[29] Rene Machhamer, Julian Altenhofer, Kai Ueding, Lars Czenkusch, Florian Stolz, Michael Harth, Markus Mattern, Aamir Latif, Sven Haab, Julian Heil, et al. 2020. Visual programmed IoT beehive monitoring for decision aid by machine learning based anomaly detection. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 1–5.

[30] Aniket Mahanti and Reda Alhajj. 2005. Visual interface for online watching of frequent itemset generation in Apriori and ECLAT. *Fourth International Conference on Machine Learning and Applications (ICMLA'05)* (2005), 6–pp.

[31] Tanmaya Mahapatra, Ilias Gerostathopoulos, Christian Prehofer, and Samarth G Gore. 2018. Graphical spark programming in iot mashup tools. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*. IEEE, 163–170.

[32] David Mason and Karthik Dave. 2017. Block-based versus flow-based programming for naive programmers. In *2017 IEEE Blocks and Beyond Workshop (B&B)*. IEEE, 25–28.

[33] Md Hasibul Masum, Tasnim Suraiya Rifat, SM Tareeq, and Hasnain Heickal. 2018. A framework for developing graphically programmable low-cost robotics kit for classroom education. In *Proceedings of the 10th International Conference on Education Technology and Computers*. 22–26.

[34] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. 2006. YALE: rapid prototyping for complex data mining tasks. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), 935–940.

[35] Giselle Nodalo, Jose Mari Sarreal III, John Valenzuela, and Jan Alliston Deja. 2019. On building design guidelines for an interactive machine learning sandbox application. In *Proceedings of the 5th International ACM In-Cooperation HCI and UX Conference*. 70–77.

[36] Youngjun Park and Youngseok Shin. 2021. Tooee: A novel scratch extension for K-12 big data and artificial intelligence education using text-based visual blocks. *IEEE Access* 9 (2021), 149630–149646.

[37] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration* 6, 3 (2017), 67.

[38] Pedro Plaza, Manuel Castro, José Manuel Sáez-López, Elio Sancristobal, Rosario Gil, Antonio Menacho, Félix García-Loro, Beatriz Quintana, Sergio Martin, Blanca Blanco, et al. 2021. Promoting computational thinking through visual block programming tools. In *2021 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1131–1136.

[39] Arvind Rao, Ayush Bihani, and Mydhili Nair. 2018. Milo: A visual programming environment for data science education. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 211–215.

[40] Tejas Reddy, Randi Williams, and Cynthia Breazeal. 2022. LevelUp–Automatic Assessment of Block-Based Machine Learning Projects for AI Education. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–8.

[41] Sergio Rojas-Galeano and Nelson Rodriguez. 2013. Goldenberry: EDA visual programming in Orange. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. 1325–1332.

[42] Anush Sankaran, Rahul Aralikatte, Senthil Mani, Shreya Khare, Naveen Panwar, and Neelamadhav Gantayat. 2017. DARVIZ: Deep Abstract Representation, Visualization, and Verification of Deep Learning Models. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. 47–50. https://doi.org/10.1109/ICSE-NIER.2017.13

[43] Jeff Sauro and James R Lewis. 2016. *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann.

[44] Allen Shen and Yuanyuan Sun. 2021. GraphicalAI: A User-Centric Approach to Develop Artificial Intelligence and Machine Learning Applications Using a Visual and Graphical Language. In *2021 4th International Conference on Data Storage and Data Engineering*. IEEE, 52–58.

[45] Weikun Shi, Zeke Dong, and Lihai Zhang. 2021. Graphical platform of intelligent algorithm development for object detection of educational drone. In *2021 China Automation Congress (CAC)*. IEEE, 6780–6784.

[46] Ben Shneiderman. 2003. Promoting universal usability with multi-layer interface design. *ACM SIGCAPH Computers and the Physically Handicapped* 73-74 (2003), 1–8.

[47] Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A Visual Programming Paradigm for Abstract Deep Learning Model Development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (Hyderabad, India) *(IndiaHCI '19)*. Association for Computing Machinery, New York, NY, USA, Article 16, 11 pages. https://doi.org/10.1145/3364183.3364202

[48] Srikanth G Tamilselvam, Naveen Panwar, Shruti Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A visual programming paradigm for abstract deep learning model development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction*. 1–11.

[49] Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A Visual Programming Paradigm for Abstract Deep Learning Model Development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (Hyderabad, India) *(IndiaHCI '19)*. Association for Computing Machinery, New York, NY, USA, Article 16, 11 pages. https://doi.org/10.1145/3364183.3364202

[50] Tiffany Tseng, Joseph Kuan-Chieh Chen, Menna Abdelrahman, Mary Beth Kery, Fred Hohman, Ashleigh Hilliard, and R Benjamin Shapiro. 2023. Collaborative Machine Learning Model Building with Families using CO-ML. In *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*. 40–51.

[51] Roxani Tsoni, Vasileios Kaldis, Irene Kapogianni, Anastasia Sakagianni, Georgios Feretzakis, and Vassilios S Verykios. 2023. A Machine Learning Pipeline Using KNIME to Predict Hospital Admission in the MIMIC-IV Database. In *2023 14th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 1–6.

[52] Serena Versino, Tommaso Turchi, and Alessio Malizia. 2024. Fostering Inexperienced User Participation in ML-based Systems Design: A Literature Review of Visual Language Tools. In *Proceedings of the 1st International Workshop on Designing and Building Hybrid Human–AI Systems (SYNERGY 2024)*. CEUR Workshop Proceedings.

[53] Randi Williams, Michael Moskal, and Peli de Halleux. 2022. ML Blocks: A Block-Based, Graphical User Interface for Creating TinyML Models. *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2022), 1–5.

[54] Chao Xie, Hua Qi, Lei Ma, and Jianjun Zhao. 2019. DeepVisual: A Visual Programming Tool for Deep Learning Systems. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 130–134. https://doi.org/10.1109/ICPC.2019.00028

[55] Cheng Xie, Haohua Qi, Lei Ma, and Jianjun Zhao. 2019. DeepVisual: A visual programming tool for deep learning systems. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 130–134.

[56] Yongchao Zhang, Yucong Wang, Hanjia Zhang, Bowen Zhu, Siwei Chen, and Dongmei Zhang. 2022. OneLabeler: A Flexible System for Building Data Labeling Tools. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–22.