Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Framework for Understanding Neural Network Component Interactions and Selection Principles, Guidelines, and Empirical Evidence

Nagwa Elmobark

eng_nagwaelmobark@yahoo.com

Mansoura University

Research Article

Keywords: Neural Networks, Component Interactions, Architecture Design, Activation Functions, Weight Initialization, Normalization, Framework

Posted Date: March 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8945550/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

Neural network architectures consist of several related components, including activation functions, weight initialization methods, normalization, optimizers, and architectures. Whereas the literature in the area has been comprehensive in describing each component separately, there is little evidence on the critical synergies and incompatibilities arising from their interactions. The paper presents a comprehensive Component Interaction Framework (CIF) that visualizes the connections among basic neural network building blocks and provides guidelines for their effective combination. We examine the effects of activation functions on weight initialization conditions, the effect of normalization strategies on optimizer choice, and how optimizer choices are affected by the architectural pattern, such as residual connections, to improve compatibility of components. A systematic analysis and experimental confirmation show that the correct pairing of different components can achieve 2-3 times faster convergence than incompatible pairs. Our system will provide novice and advanced researchers with practical decision trees and compatibility matrices to construct effective neural architectures. This publication bridges the gap between theoretical knowledge and practical architectural engineering and provides a coherent view of neural network design that focuses not on optimizing components in isolation but on overall reasoning.

1. Introduction

Designing neural networks requires selecting several interdependent components simultaneously, whose interactions are crucial to model performance. To construct a vision transformer, a practitioner needs to select activation functions, initialization schemes, normalization layers, and optimizers, which interact in complex ways to influence training dynamics and accelerate convergence [1]. Although deep learning architectures are becoming more advanced, whether convolutional networks or transformers [2], [3], the literature reports on these elements in isolation, as autonomous modules, rather than as interacting systems. Such fragmentation leaves scholars without a systematic means to understand how component decisions interact; instead, they must either resort to haphazard experimentation or simply rely on pattern designs whose principles of operation are opaque.

The empirical implications of this knowledge gap are huge. An example of a researcher applying an image classification network is to use ReLU activation (computationally efficient and commonly used [4]) with Xavier initialization (mathematically principled to preserve variance [5]). Although each of these decisions individually seems good, the network is unstable at gradient gradients and it takes the network 67 epochs to attain 90 % training accuracy- almost three times as many epochs as the 23 epochs attained with appropriate He initiation [6]. The underlying cause is component incompatibility: the ReLU's asymmetric output distribution, in which negative inputs yield zero output, is a fundamental failure of the assumption of symmetric activations in Xavier initialization, which posits approximately unit responses near zero. The resulting mismatch leads to a collapse of variance in the deep layers, which degrades gradient flow and slows convergence. Although these interactions are of paramount importance to performance, the literature provides them with little systematic treatment. Practitioners face the same

difficulties when using normalization methods and optimizers, architecture schemes with initialization methods, or activation functions with residual connections. Both combinations introduce minor dependencies that may accelerate training, introduce instability, or silently degrade final performance.

The existing literature on component selection comprises three major studies, each of which is highly limited. First, there is much published on single elements, such as activation functions [4], [7], [8], and initialization procedures [5], [6], [9], normalization procedures [10]- [12] and optimizers [13]- [15], but these papers only analyze each aspect individually and not the interactions. Second, automated architecture search systems, such as Neural Architecture Search (NAS) [16] and AutoML systems [17], may find good combinations of components by extensive computational search, but do not give any explanations of why these combinations are successful and why others are not, which limits their use in understanding and generalizing. Third, design pattern libraries and best-practice books [18] empirically describe proven architectures but often do not explain the interaction principles underlying these combinations. Most importantly, there is no single framework that describes the mechanistic interactions among components, how activation function properties constrain the requirements of an initiation process, how normalization strategies alter loss landscapes to influence the efficiency of optimizer's and how architectural structures such as residual connections alter the compatibility of other components.

In this paper, these gaps will be addressed by outlining a systematic approach to understanding and using neural network components, given their synergistic interactions: the Component Interaction Framework (CIF). Instead of proposing new architectures or components, we establish a systematic approach to reasoning about components and their interactions. The framework classifies the elements of a neural network into four layers that are linked together: data foundation, core computation, stabilization, optimization, and the architectural patterns. It has been found that important interaction principles govern their compatibility. We offer practical decision support tools such as: (1) interaction maps of documentation of dependencies between category of components (2) compatibility matrices of quantification of effectiveness of certain combinations of components based on theoretical and empirical analysis (3) decision flowcharts of selecting components based on task requirements, network depth, and resource constraint (4) experimental confirmation of CIFAR-10 showing that correct combination of components can converge 23X faster than incorrect combination as well as improve final accuracy by up to 5 % points.

Fourfold are our particular contributions. We begin by providing a complete taxonomy of component interactions in contemporary supervised learning architectures, with three types of interactions, namely, synergistic, neutral, and antagonistic, and describe the properties of each of them in convolutional networks, transformers, and multilayer perceptrons. Second, we present practical recommendations for mapping component selection to the architecture, with 22 rules covering activation functions, initialization, normalization, optimizers, and the integration of architectural patterns. Third, we test the structure experimentally through systematic ablation experiments that compare convergence rates and final performance across combinations of components, thereby quantifying the effect of compatibility

on training efficiency. Fourth, we provide decision-support formats, such as compatibility matrices and selection flowcharts, tailored for novice and intermediate researchers who may not be quite familiar with component interactions. The framework is centered on core, general, widely used elements that underlie modern architectures, providing a foundation that can readily be extended to newer techniques, such as activation functions or normalization variants.

We limit this paper to supervised learning architectures in computer vision and natural language processing, including convolutional neural networks (CNNs), transformer architectures, and multilayer perceptrons (MLPs). Although the interaction principles, which are variance preservation, gradient flow maintenance, and loss landscape geometry, are generalizable, we are not dealing with reinforcement learning or unsupervised learning algorithms, which have different interaction principles given their different objective functions and training dynamics. Although we focus on conventional training schedules rather than specific applications such as few-shot learning, continual learning, or highly resource-constrained deployment, the framework's principles can also be applied to component selection for these applications.

The rest of this paper is structured as follows. Section 2 is a literature review of the background of neural network components and architecture design frameworks. Section 3 identifies research gaps in knowledge of component interactions. Part 4 describes the Component Interaction Framework, its layered structure, and its key interaction principles. Section 5 provides detailed practical instructions for selecting components across various architectural contexts. Section 6 confirms the framework using CIFAR-10 experiments, showing that quantitative performance varies across component combinations. Section 7 discusses the theoretical foundation, cross-domain generalization, and the limitations of structural approaches. Section 8 outlines future work. Section 9 summarizes the major conclusions and implications of the neural network design practice.

2. Background and Related Work

This section discusses the basic neural network elements that will be central to our framework and analyzes existing methods for component selection and architecture design. We divide the discussion into two sections: the first one is where we survey the individual component categories (Sections 2.1-2.5), their properties, and the interaction-relevant characteristics that have been previously considered unproblematic, and the second is where we critically examine existing methods of component selection and architecture design (Section 2.6), where our contribution falls relative to these methods.

2.1 Activation Functions

Activation functions provide much-needed nonlinearity, allowing neural networks to model intricate patterns beyond linear transformations. The gradient flow, training stability, and convergence speed depend heavily on the activation function used; yet most prior work evaluates these properties separately, without examining how their statistical properties affect compatible initialization techniques or how they interact with normalization layers.

ReLU family (ReLU [4], Leaky ReLU [19], PReLU [20]) is computationally efficient due to the use of simple thresholding and overcomes vanishing gradients by maintaining gradient magnitude in linear areas. Nonetheless, the asymmetric output distribution of ReLU, i.e., producing zero for any negative input, makes the statistical properties of its outputs inherently distinct from those of symmetric activations, with crucial implications for initialization algorithms, as we discuss in Section 4. Dying ReLU problem, in which neurons permanently deactivate, inspires alternatives such as Leaky ReLU that allow small negative gradients [19].

The sigmoid and hyperbolic tangent (tanh) families provide outputs constrained to (0,1) and (-1,1), respectively, and are therefore applicable to tasks such as binary classification output layers or constrained regression problems [21]. Both functions are symmetric about their midpoints and have relatively unit derivatives at zero, which affects compatible initialization schemes. However, they exhibit vanishing gradients in deep networks due to saturation regions, where gradients tend to zero out when input magnitudes are large, including only shallow networks or their output layers [22].

Probabilistic interpretations and continuous differentiability are implemented in modern smooth activation functions, such as GELU [7] and Swish [8]. GELU, popular in transformer architectures [3], is the cumulative distribution function that is weighted, which offers smooth nonlinearity, which empirically is an optimization benefit in very deep networks. Swish (or SiLU) multiplies the input with its sigmoid and attains self-gating behaviour. These functions exhibit lower sensitivity to initialization choice than ReLU, but the mechanisms underlying this resistance remain incompletely understood [23].

Gap identified: Although properties of individual activation, such as the computational cost, gradient behaviour, and approximation capacity, have been described extensively, there is limited system-level analysis of how such properties influence compatibility with other components. An example is the interaction between the symmetry of activation and the symmetry of requirements between activation and activation initialization, or the interaction between activation smoothness and the placement of the normalization layer, which is often treated as a matter of empirical observation rather than a subject of coherent research.

2.2 Weight Initialization

Proper initialization eliminates pathological training dynamics, such as vanishing and exploding gradients, in the initial stages of training, where the magnitude of weights significantly affects gradient flow [5], [6]. Initialization strategies are designed to preserve activation and gradient variance across layers, but they implicitly aim to make the activation functions compatible, even when they are not explicitly specified. Random initialization provides baseline approaches, typically drawing weights from normal or uniform distributions with manually tuned variance [24]. While simple, random initialization lacks theoretical grounding and often requires problem-specific tuning, making it unsuitable for general architectural design principles.

Xavier (Glorot) initialization [5] preserves variance between layers by scaling weights based on both fan-in and fan-out, and calculates optimal variance under the condition of both linear activations and nonlinearities with a derivative of about a unit around zero. This is true of tanh and sigmoid activations, but essentially false of ReLU, which gives zero values at half its range of input. Nevertheless, the interaction between Xavier initialization and the ReLU activation function, despite this incompatibility, is not sufficiently emphasized in introductory literature, which is a common source of implementation errors.

His (Kaiming) initialization [6] scales the variance when using ReLU activation, exploiting the zero-negative region by doubling the variance relative to Xavier. This modification preserves signal variance using a ReLU activation, thereby averting the collapse of variance that causes vanishing gradients. The derivation of the initialization is a direct statistical modeling of the ReLU properties, an example of how initialization should conform to the properties of activations, which our framework generalizes.

Orthogonal initialization is A method used to maintain gradient norms by initializing weight matrices using orthogonal matrices, which is found to be especially useful in recurrent architectures where gradients flow over time [25]. LSUV (Layer-Sequential Unit-Variance) [26] offers data-driven initialisation by using forward passes to empirically determine and compensate for layer-wise variance, and is robust in the event of theoretical failure.

Gap identified: The Initialization literature typically treats the choice of activation function as context-dependent rather than a design variable, and focuses on initializing with ReLUs rather than on how activation choice constrains the choice of initialization policies. This framing blurs the two-way relationship: activation properties can dictate what is required to initialize the system, and initialization capabilities can, in turn, affect the choice of activation under resource-constrained conditions.

2.3 Normalization Techniques

Normalization layers stabilize training by normalizing activation distributions, reducing internal covariate shift, and smoothing the loss landscape [10]. Various normalization schemes normalize in different dimensions' batch, layer, or channel groups, and generate different statistical properties with large consequences on optimizer selection and batch size scales.

Activations are normalized independently across the batch dimension, with mean and variance computed over the minibatch used. BatchNorm enables faster convergence and higher learning rates by insulating the network from initialization and input distribution changes [27]. Nonetheless, this dependence on the batch dimension makes batch size dependent on model performance: small batches yield noisy statistics that cause training instability, and, in general, batch sizes must be at least 16 [28]. Recent work by Santurkar et al. [29] shows that the main advantage of BatchNorm is due to loss-landscape smoothing, rather than covariate-shift reduction, a geometric phenomenon that can have significant implications for optimiser selection, as discussed in Section 4.

Normalization is performed on the feature dimension rather than the batch dimension, and the statistics are computed separately for each sample [11]. Such independence across batches makes LayerNorm vital for models with variable batch sizes and for recurrent models, where batch normalisation is problematic [30]. Transformer architectures are dominated by LayerNorm [3], which, in combination with self-attention mechanisms and position-wise feedforward layers, yields training dynamics that differ in character from those of BatchNorm in convolutional networks.

GroupNormalization (GroupNorm) [12] splits channels into groups and normalizes each group, a midpoint between BatchNorm and LayerNorm. GroupNorm can be applied in batches of any size; it is particularly useful for small-batch applications, such as object detection and video recognition [12]. The other variants, such as Instance Normalization, address domain requirements [31].

Gap identified: Normalization research primarily considers domain-specific properties of individual techniques and their performance. Nonetheless, normalization layers fundamentally alter the geometry of loss landscapes [29], creating new optimizer-favouring geometries: BatchNorm smooths first-order approaches such as SGD, whereas LayerNorm supports adaptive approaches such as Adam. Such optimizer interactions are not studied with much systematic attention although they are often of practical significance.

2.4 Optimizers

The network parameters are updated by optimization algorithms that aim to minimize loss functions, and the choice of algorithm plays a significant role in the speed of convergence, final performance, and training stability [32]. The design of optimizers has moved beyond basic gradient descent to more advanced adaptive algorithms, but a systematic analysis of how optimization behavior depends on the architectural building blocks has not been achieved.

Gradient accumulation with momentum, Stochastic Gradient Descent (SGD) [13], introduces velocity terms that aggregate gradients over the training process, helping overcome local minima and accelerating convergence in ravine-like loss functions. Although it is simple, SGD with momentum achieves competitive or better performance than adaptive methods when using appropriate learning rate schedules and normalization layers [33]. This observation implies that we can use components of architecture (especially normalization) in order to offset the simplicity of optimizers- an interaction that we model in our framework.

RMSProp [34] and its refinement, Adam [14], are adaptive per-parameter learning rates based on moving averages of squared gradients (RMSProp) or first- and second-moment gradients (Adam). Such adaptive techniques make them less sensitive to the choice of learning rate, and in early training, they tend to converge quickly. Nonetheless, the latter study identifies gaps in generalization between Adam and SGD in certain areas [35], and the tuning requirements of adaptive approaches are determined by hyperparameter choices, such as β^1 and $\beta 2$.

AdamW [15] separates weight decay with gradient-based updates, which is one inherent problem with the weight decay that was originally implemented in Adam. This decoupling also yields stronger regularization and typically improves final performance, particularly in large-scale transformer training [36]. The AdamW formulation describes the interaction between the optimization design and the regularization strategies, which our framework also considers.

More refinements have been proposed by recent optimizers, such as Lookahead [37], RAdam [38], and Sophia [39], but their use is less common than that of the methods mentioned above.

Gap in knowledge: Published literature on optimizers has generally evaluated performance on fixed architectures using standard components, obscuring the dependence of optimiser performance on normalization choices, the smoothness of activation functions, or the geometry of loss landscapes shaped by architectural templates. An example of an interaction getting inappropriate attention is the finding that SGD is competitive with BatchNorm but not without [29].

2.5 Architectural Patterns

In addition to the choices of each individual component, architectural designs such as residual connections, attention mechanisms, and positional encodings fundamentally alter the characteristics of information flow and gradient propagation and affect the requirements of other components.

Through residual connections [2], it is possible to train extremely deep networks (100 or more layers) by creating gradient highways via skip connections, avoiding nonlinear transformations. Residual connections modify the requirements on initializing a network - networks with residuals admit easier initialization schemes and accumulate variance in a different way than do purely sequential networks [40]. The relationship between residual connections and the placement of normalization layers (pre-activation or post-activation) has a substantial impact on training stability, and different configurations prefer different initialization and normalization options [41].

The selective processing of information through learning to weight inputs provides attention mechanisms [42] that store long-range relationships within a sequence. Position-invariance of self-attention requires positional encodings [3], which is a dependency between architecture pattern (attention) and component demands (position information). The parallel processing of attention heads in multi-head attention is integrated with normalization layer placement and dropout techniques in a manner that influences optimal hyperparameter choices [43].

Positional encodings provide positional information to position-invariant models, such as transformers [3]. Model capacity, sequence-length generalization, and downstream parameter initialization also interact with the option of using sinusoidal encodings (deterministic, with no learned parameters) or learned embedding.

Gap identified: The literature on architectural patterns captures the relative merits of each pattern (residual connections are identified, dependencies are discovered and captured), but seldom

institutionalizes how patterns alter the requirements on basic components. As an example, the effect of residual connections on the accumulation of gradient variances has explicit consequences on initialization scaling, but is not explicitly described in the majority of residual network works [2]

2.6 Related Frameworks and Approaches

Having viewed and scrutinized the individual elements, now we are going to study actual methods of component selection and architecture designing, placing our framework among four main paradigms, including automated architecture search, theoretical analysis, empirical design pattern, and interaction-centered research.

2.6.1 Automated Architecture Search

Neural Architecture Search (NAS) architectures [16], [44]– [46] can be designed through methods that search architecture components [combinations] by learning through reinforcement learning, evolutionary search, or gradient-based optimization. Such methods have yielded architectures with state-of-the-art performance [16], [44], demonstrating that effective combinations of components can be derived from manually designed patterns. Nevertheless, NAS methods have fatal limitations to component interactions: (1) they generally require enormous amounts of computational power (thousands of GPU hours) to be feasible at all, which is not feasible in typical research projects [47], (2) the architectures identified by search parent to themselves do not give a meaningful explanation of how and why particular components combination works, which impedes generalisation to new domains and constraints [48], (3) even the search spaces must be defined by humans, which already requires prior knowledge of reasonable combinations of components the very thing we are trying to formalise.

AutoML systems [17], [49] can automate the entire machine learning workflow, including preprocessing, architecture selection, and hyperparameter tuning. Although they are useful when practitioners need to have a performant model but do not need to understand the underlying principles of interaction, they will not give information on the principle of interaction. Recent explainable NAS effort [50] works seek to explain discovered architectures but in terms of architecture motifs as opposed to the mechanism of component interaction.

Limitation: Automated search finds effective combinations but fails to explain principles of interaction therefore restricting generalization and understanding.

2.6.2 Theoretical Analysis

Theoretical works examine particular issues of the dynamics of neural network training concerning components relations. Infinite-width limits Infinite-width limits [51] describe the dynamics of training and give insight into the behavior of initialization and learning rate scaling, but tend to make simplifying assumptions (e.g. particular activation functions, infinite width) that are not easily applied in practice. Training in the lazy regime where the weights change only a little over time is studied using Neural

Tangent Kernel (NTK) theory [52], and provides some theoretical guarantees, but again can be quite different than practical finite-width deep learning behaviour where weights change significantly [53].

Signal propagation analysis [5], [6] examines the propagation of activation variance in a network during forward and backward propagation and directly influences the initial design of activities. This body of work exemplifies theoretical analysis that has been effective in informing practical component selection, but generally treats the initialization and activation factors in isolation, with no systematic generalization to normalization or optimizer interactions.

Loss landscape analysis [29], [54] describes the behavior of architectural design in terms of optimization difficulty. The major advantage of BatchNorm, as Santurkar et al. [29] have shown, is due to loss-landscape smoothing, not to alleviating internal covariate shift, which provides empirical support for the theoretical claim that BatchNorm allows SGD to optimize effectively. This publication is an exceptional, methodical exploration of part interaction (normalization-optimizer), but it focuses on a single pairing rather than on the creation of an overall structure.

Limit: Theoretical work provides a profound understanding of specific interactions, but it typically analyzes small case studies (e.g., individual activations, infinite-width limits, specific component pairs) and does not necessarily offer high-level practical advice.

2.6.3 Empirical Design Patterns and Best Practices

There is extensive literature describing successful architectural patterns and component combinations based on empirical evidence [18], [55], [56]. These are practitioner documentation, pre-trained neural network libraries with default settings, and application-specific advice (e.g., "apply BatchNorm to CNNs, LayerNorm to Transformers). These resources are of immediate practical value, but they tend to offer recommendations in the form of rules without providing explanations that would allow flexibility for new situations or constraints.

Survey papers [57], [58] also provide an extensive overview of architectural families and component choices, but generally organize information by component type rather than by interaction principles. Recent multidimensional analysis frameworks [56] have explored trade-offs in model selection; however, they are overly focused on algorithm-level comparisons rather than on detailed component interactions at the architectural level. For example, an activation function survey could brevitate the computational cost and gradient behavior of variants of ReLU, but not systematically consider how the statistical behavior of each variant limits the choice of initializations.

The implicit encoding of component interaction knowledge in successful settings (e.g., GELU + LayerNorm + AdamW) in research on transfer learning and pretrained models [59] remains poorly understood, beyond empirical performance reports.

Limitations: Empirical patterns provide useful starting points but do not offer explanatory accounts of when patterns generalize and when alternative combinations may be preferable under certain

constraints.

2.6.4 Interaction-Focused Studies

Component interaction is explicitly studied in a small but growing body of work, yet remains sporadically covered. Santurkar et al. [29] study interactions between BatchNorm-optimizers, showing that BatchNorm smooths loss landscapes, which explains its synergy with SGD. The paper by Zhang et al. [40] investigates the influence of residual connections on initialisation requirements, finding that residuals reduce the sensitivity to initialisation scale. Hanin and Rolnick [61] study the depth of interactions at initialization and obtain initialization schemes for very big networks. These papers present strong interaction analyses but emphasize particular pairings (normalization-optimizer, architecture-initialization, initialization-depth) rather than providing informative frameworks.

Recent papers suggest that certain components should be partially integrated. For example, Fixup initialization [62] proposes initialization plans that avoid normalization in residual networks, demonstrating that initialization can compensate for the lack of normalization. Sharpness-Aware Minimization [63] is a form of optimization that makes optimization more generalizable; its performance varies with the interactions between optimizer behavior and loss-landscape geometry shaped by normalization.

Limitation: Although such works provide essential information about particular interactions, there is no comprehensive framework that consolidates these findings into a coherent perspective, enabling the systematic selection of elements across different architectural settings.

2.6.5 Positioning Our Contribution

Our Component Interaction Framework (CIF) has four major distinctions with existing approaches in the following four dimensions:

1. Interpretability vs. Automation: In comparison to NAS/AutoML, we do not use black box search, but knowledge transfers and adaptation with interpretable principles of why component combinations will work or not.

2. Breadth vs. Depth: In contrast to theoretical research, which concentrates on the analysis of particular interactions, we offer a holistic coverage of the basic building blocks (activation, initialization, normalization, optimizer, architectural patterns), allowing one to design architecture holistically.

3. Explanation vs. Documentation: We base our recommendations on mechanistic knowledge of component properties (statistical distributions, gradient flow, geometry of loss landscapes), unlike empirical pattern catalogs, meaning that practitioners can think about new situations rather than the patterns they have recorded.

4. Systematic Framework vs. Fragmented Findings: Contrary to research studies of interaction scattered, we present the findings in a coherent four-layer framework with defined interaction principles, compatibility matrices, and decision support tools, and offer a structured methodology of component selection.

The framework integrates theoretical (e.g., principles of variance preservation), empirical (e.g., the effectiveness of BatchNorm and SGD), and systematic (Section 6) research into guidelines that can be implemented by researchers at different experience levels. Through the interaction approach, we bridge the gap between identifying the components and combining them successfully to form an entire entity.

3. Research Gap

The literature reviewed in Section 2 demonstrates significant advances in understanding individual aspects of neural networks. Nevertheless, there are still critical gaps in the systematic treatment of the interactions between the components - dependency, synergy and incompetency that arise when components are brought together in full architectures.

3.1 Gap 1: Limited Interaction Documentation

Most of the current researches study components separately. The effects of normalization on training dynamics are investigated in batchNorm studies [10], which do not explicitly study the effect of normalization on the dependence on the choice of activation function or type of optimizer. Equally, the activation set of researches [4], [7], [8] characterizes the properties of individual functions, such as computational cost, gradient behavior, saturation properties, but it does not discuss the limitations of the properties on incompatible initiation strategies. This separation masks important properties: e.g. the loss landscape smoothing of BatchNorm [29] marks its advantage over first-order optimizers (SGD) or adaptive ones (Adam), or the asymmetric output distribution of ReLU makes the symmetric activation assumption of Xavier initialization [5], [6] fundamentally false.

The small number of studies that have investigated interactions specifically [29], [40], [61] are also valuable but are singularly focused, and do not attempt to formulate systematic interaction principles that can be applied to different categories of components. Practitioners, in turn, do not have an extensive principle in how to reason about combinations of various components that have never been reported in the scattered literature.

3.2 Gap 2: Absence of Practical Compatibility Guidelines

Although there is extensive documentation on components, there are no systematic compatibility guidelines. The specific questions that researchers ask when designing architecture include: Which form of initialization is appropriate with GELU activation? What is the impact of GroupNorm choice on the choice of optimizer with a batch size of 16? Does the presence of residual connection alter normalization layer condition? They are not consistently covered in the literature, and solutions to the

questions are distributed across the literature on specific domains, practical guidelines, and empirical evidence, rather than in a single systematic examination.

Current literature offers highly specific recommendations tied to specific architectures (e.g., ResNet-50 should use BatchNorm with SGD momentum 0.9) or overly generic advice (e.g., select the right activation). The intermediate level of systematic principles that justify why some combinations are effective and allow practitioners to reason about new situations is mostly in place. This incompleteness compels researchers to experiment through trial and error, which is both computationally inefficient and may lead to suboptimal settings.

3.3 Gap 3: Insufficient Beginner-Accessible Frameworks

The existing sources either presuppose a high level of background knowledge or provide cookbook-style lists without explanation. Further theoretical development [51]–[53] On the other hand, the successful patterns are captured by practitioner guides [18], [55], [56] but they seldom specify the underlying mechanism of interaction thus making it difficult to transfer knowledge to new situations.

Novice scientists and amateur researchers must have frameworks that justify why some combinations work and the mechanistic associations between the properties of components and the outcomes of interactions. Examples To explain why ReLU+Xavier is an unsatisfactory model, the concept that zero-negative area in ReLU breaches the assumptions of the Xavierian symmetric activation hypothesis can be applied to admit to other asymmetric activations, but not memorising the fact that ReLU+Xavier is bad. This explanatory disjunction is especially experienced by new entrants into the component world of growing activities and tasks, where there are new activation functions, normalization forms and optimizer forms being added all the time.

3.4 Gap 4: Limited Quantitative Interaction Analysis

Although there are qualitative recommendations (use He initialization with ReLU, BatchNorm lets you use higher learning rates) the interaction effects have not been quantitatively studied. To what extent does the inappropriate use of activation-initialization pairing impair convergence speed? Assuming that there is a difference in performance between normalization-optimizer compatibility? What are the most important interactions to get the final accuracy versus training efficiency? These are questions which need to be systematically investigated empirically with comparison of components combinations under controlled conditions, but most of the literature focuses on the analysis of components individually or presents the results of single fixed configurations.

The small number of quantitative studies [29] do show the importance of interaction, as BatchNorm allows Adam to achieve Adam-theoretically advantageous performance, but only with a particular combination. Extensive quantitative characterization in interaction types would allow practitioners to make some alternatives according to compatibility considerations: in case an activation-initialization

mismatch slows convergence three times and a normalization-optimizer interaction accelerates 20 %, practitioners can devote optimization efforts to that interaction.

3.5 Gap 5: Lack of Decision Support Tools

During architecture design, researchers are not provided with any practical tools to guide them in selecting the component. Though search is automated by NAS methods [16], [44] -[46], they need large amounts of computational resources and do not give interpretable explanations of found combinations. Practitioners require tools of intermediate complexity decision trees, compatibility matrices, selection flowcharts that are used to choose components according to the needs of the task, architecture constraints and the availability of resources and provides the reasoning behind such choices.

These tools must accommodate the difference in expertise: novices should be provided with simple starting points, known pitfalls they must avoid, intermediaries should be provided with systematic principles by which they can think about new combinations, experts should be provided with full-fledged interaction characterization to aid optimization choices. The existing literature does not offer the much-needed unified decision frameworks or tools along with the component selection knowledge arranged at the right abstraction level.

3.6 Summary of Research Gaps

Table 1: Identified Gaps, Practical Implications, and Corresponding Contributions of This Framework

Gap	Current State	Impact	Our Contribution
Interaction Documentation	Components analyzed in isolation	Practitioners are unaware of dependencies	Systematic interaction taxonomy (Section 4)
Compatibility Guidelines	Scattered, domain-specific advice	Trial-and-error experimentation required	Compatibility matrices and decision rules (Sections 4–5)
Beginner Accessibility	Either too theoretical or cookbook recipes	Knowledge doesn't transfer to novel scenarios	Explanatory framework with mechanistic rationale (Section 4)
Quantitative Analysis	Limited empirical interaction studies	Unknown relative importance of interactions	Systematic ablation studies (Section 6)
Decision Tools	Only automated (NAS) or manual search	No intermediate-complexity guidance	Decision flowcharts and selection guidelines (Section 5)

Taken together, these gaps pose a major issue: practitioners know a great deal about each individual element but have no systematic way of integrating them to benefit. Our Component Interaction

Framework helps address these gaps by offering interpretable interaction principles, quantitative validation, and decision-support tools accessible to researchers at all levels of experience.

Our Contribution: We address these gaps with a systematic framework for documenting interactions among components, which is quantitatively validated and provides practical decision support for researchers at different levels of experience.

4. Proposed Framework: Component Interaction Framework (CIF)

This section introduces the Component Interaction Framework (CIF), a systematic approach to understanding and selecting neural network components based on their interrelations. The model divides elements into four interaction layers and establishes rules within each layer to determine their compatibility.

4.1 Framework Architecture

CIF structures neural network components into four hierarchical layers that reflect their roles in the computational pipeline (Figure 1):

Layer 1: Data Foundation - Input characteristics (dimensionality, distribution, modality) and preprocessing operations that establish the data representation entering the network.

Layer 2: Core Computation - Fundamental computational elements including learnable parameters (weights, biases), initialization strategies determining initial parameter values, and activation functions introducing nonlinearity.

Layer 3: Stabilization and Optimization - Components controlling training dynamics, including normalization techniques stabilizing activation distributions, loss functions defining optimization objectives, and optimizers updating parameters.

Layer 4: Architectural Patterns - Structural components at the high level, such as residual connections through which gradient flow can be performed in deep networks, attention systems through which selective information processing can be performed, and positional encodings through which sequence order information can be injected.

All layers have hierarchical dependencies, such that Layer 2 decisions constrain Layer 3, and Layer 4 patterns alter the requirements of the lower layers. For example, residual connections (Layer 4) reduce sensitivity to initialization (Layer 2) and alter the placement of normalization (Layer 3).

4.2 Key Interaction Principles

The framework determines four basic principles of compatibility of components:

Principle 1: Activation-Initialization Coupling- The statistical properties of activation functions can be used to identify an appropriate initialisation strategy. In order to have symmetric activations (tanh, sigmoid) with a roughly unit derivative around 0, schemes that are variance-preserving, such as Xavier initialization, are necessary. Asymmetric activations (E.g., ReLU variants) with negative regions that are zero require gain-compensated schemes, such as He initialization, to account for the loss of effective dimensionality. Such a connection arises from the propagation of variance analysis: proper initialization ensures that the activation variance of layers remains constant, preventing vanishing or exploding signals.

Principle 2: Interaction of normalization-Optimizers - Techniques of normalization transform geometry on loss landscapes, differentially impacting the efficiency of optimizers. That is smooths the landscape of the first-order method (SGD) so that it can effectively navigate the landscape, even without adaptive learning rates. LayerNorm yields various geometric properties, and adaptive techniques (e.g., Adam) more effectively leverage per-parameter learning-rate adjustments. The above interaction shows that normalization not only alters the network function but also, in optimal cases, the optimization problem itself.

Principle 3: Architecture-Component Modification - Architectural patterns are patterns that radically change component requirements. Gradient highways offered by residual connections yield a level of gradient highways that is insensitive to initialization; networks with residuals can support simpler network initialization schemes. Position information is needed by attention mechanisms, which means that position-invariant architectures need positional encodings. These changes propagate: the effect of residual connections on the optimal normalization point (pre-activation or post-activation) is also present.

Principle 4: Depth-Dependent Selection - Effects of component interaction scale with network depth. Deep networks (more than 50 layers) are sensitive to the coordination among the initialisation, normalisation, and architectural patterns to preserve gradient flow. The tolerances of shallow networks (less than 10 layers) are to mismatches, which would cause deep networks to destabilize. This depth dependence reflects cumulative changes in variance and in gradient transformations across a large number of layers.

4.3 Interaction Categories

Component interactions fall into three categories based on their combined effect (Figure 2):

Synergistic Interactions - Elements that increase the performance of others in a way that goes beyond the contribution of one. Examples: He initialization + ReLU activation preserves variance regardless of ReLU asymmetry; LayerNorm + Adam optimization leverages the geometry of the landscape and adaptive rates; residual connections + deep architectures enable training beyond conventional depths.

Neutral Interactions - Components that do not have a strong association with one another. Examples: the majority of normalization-loss function pairs differ in different directions (stability of training versus training objective); positional encoding-optimizer selection has different orthogonal issues.

Antagonistic Interactions - The elements that conflict with each other in terms of functioning, lowering the joint performance. Examples include Xavier randomization + ReLU activation leading to variance collapse, BatchNorm + small batches (smaller than 4) leading to noisy statistics, and large learning rates + saturating activations leading to gradient instability.

4.4 Activation-Initialization Compatibility Analysis

Systematic compatibility analysis between the activation functions and the strategies of initializing is given in Figure 2 regarding the propagation principles of the theoretical variance based on the empirical validation. The trends are evident in the matrix:

ReLU family needs He init to work best. Output asymmetry (zero on negative inputs) reduces effective network capacity by 50 percent and doubles the required initialisation variance (to ensure signal propagation) for ReLUs. The symmetric activations of Xavier initialization lead to a collapse of the variance in ReLU networks, resulting in vanishing gradients and slow convergence. The decrease in sensitivity of Leaky ReLU is slightly smaller than that of the negative slope because small negative slopes retain some gradient flow.

Tanh and sigmoid (symmetric) are the best activations with Xavier. These functions ensure that the derivative of the unit shape around zero and the symmetric distribution of output, as is assumed by Xavier. His initialization introduces excessive variance relative to symmetric activations, which may be unstable with respect to the gradient during initial training.

More recent smooth activations (GELU, Swish, ELU) are also resistant to initialisation schemes. Their stable training via Xavier and He initialization is enabled by their continuous differentiability and lower saturation, but He is preferable because of the asymmetric nature of its outputs, similar to that of ReLU. This strength is why they are increasingly used in stationary architectures, where sensitivity during initialisation causes problems during deployment.

LSUV startup offers universal compatibility with data-driven variations correction. LSUV allows any activation function via forward passes and empirical tuning of the layer-wise variance. Nevertheless, this method requires additional computation and information retrieval at startup, which is inapplicable when the startup criteria are very strict or the privacy requirements are high.

To provide guidance for choosing the appropriate initialization scheme, Table X presents the suggested correspondence between popular activation functions and their initialization strategies. This table transforms the theoretical concepts discussed in this section into actionable principles by translating the principle of variance propagation into design guidelines. The guidelines are designed to help

beginners make informed architectural decisions that ensure consistent signal flow and reliable convergence during training.

Table 2: Recommended Initialization Strategies for Common Activation Functions

Activation Function	Recommended Initialization Method	Rationale (Theoretical Justification)
ReLU	**He Initialization**	ReLU suppresses negative activations, effectively halving signal variance. The initialization compensates by increasing initial weight variance, preserving forward/backward signal propagation, and mitigating vanishing gradients.
Leaky ReLU	**He Initialization**	The small negative slope retains partial gradient flow, but variance reduction still occurs; He initialization provides the optimal variance scaling for stable training.
ELU	**He Initialization**	ELU exhibits asymmetric activation behavior similar to ReLU-based families; increased variance scaling maintains stable gradients during early training.
GELU	**He Initialization (preferred)** or **Xavier**	Smooth curvature and reduced saturation make GELU robust across initialization schemes. Slight asymmetry favors He, though Xavier remains viable.
Swish	**He Initialization (preferred)** or **Xavier**	Swish maintains nonlinearity with smooth transitions and accommodates multiple initialization methods, thereby improving consistency in deep architectures.
tanh	**Xavier Initialization**	tanh is symmetric around zero and maintains near-unit derivatives for small inputs. Xavier matches these assumptions by balancing variance across layers.
sigmoid	**Xavier Initialization**	Symmetric activation with saturating boundaries; Xavier maintains controlled variance that reduces early-layer gradient decay.
Softsign / Softplus	**Xavier Initialization**	Smooth, symmetric activations benefit from a balanced distribution of variance; Xavier prevents excessive variance growth.
All activation functions	**LSUV Initialization (universal)**	LSUV empirically normalizes layerwise activations via data-driven variance correction, ensuring compatibility across activation functions.

4.5 Decision Framework

At the first decision level, task requirements determine the baseline component configuration.

Computer vision architectures are generally based on ReLU activations, He initialization, and BatchNorm and are indicative of the consistency of these elements in convolutional contexts and their suitability with the large batch sizes typical of vision pipelines. Conversely, the smoothness of activations (e.g. GELU, Swish) in the context of LayerNorm, normalization mechanism of which does not rely on batch

statistics, is an advantage to sequence modeling architectures, which frequently have variable-length inputs and are batch-independent in dynamics. General-purpose MLPs are not special-purpose, and so they can be configured to allow any downstream.

The second level incorporates architectural depth as a structural modifier.

In general, shallow networks do not require much normalization or standardization to work with. With depth, normalization positioning is highly important to stability in gradient, and intermediate depth networks demand careful normalization techniques to prevent signal propagation degradation. Deep networks, namely those with more than tens of layers, require residual connections as well as suitable normalization because, as the empirical and theoretical experience shows, both of these mechanisms alone do not support effective gradient flow in very deep networks.

The final decision stage integrates computational resource constraints.

BD uses steady and accurate statistics when batch sizes are large. Medium batch setups prefer GroupNorm, that is sale to the effect of batch changes without maintaining constant variance estimates. Small-batch or micro-batch LayerNorm or other batch-independent variants are needed to prevent noisy updates and keep training stable.

In general, the framework makes the design space dependent: the type of the task defines the initial design space, the architectural depth defines the stability requirements, and the availability of the resources restricts the available normalisation mechanisms. The framework allows practitioners to build to the principles of interaction by suggesting compatibility-preserving options where conflicts arise (e.g. LayerNorm) with small-batch vision models.

4.6 Framework Validation Approach

Section 6 empirically supports the frameworks' predictions through systematic ablation experiments on image classification in CIFAR-10. Convergence speed differences in activation-initialisation combinations and normalisation-optimiser pairs are experimentally measured and show that compatible combinations of frameworks that are recommended by their builders are 2-3 times faster to converge than incompatible pairs found in Figure 2.

5. Practical Guidelines for Component Selection

The section provides practical rules for selecting and combining neural network elements. These regulations translate the principles of interaction in Section 4 into specific recommendations, organised by component category.

5.1 Activation Function Selection

Guideline 1: Default to ReLU for computer vision tasks

ReLU is computationally efficient and has been shown to be successful in convolutional models. Combine with the He initialisation to preserve the variance in the asymmetric output distribution. Applications of CNNs include processing images, videos, or spatial data when translation invariance and local feature detection are prevalent.

Guideline 2: Use GELU or Swish for transformers and sequence models

Gradient flow is useful when the architecture is smooth in a deep sequential. The probabilistic interpretation of GELU and self-gating Swish enhance the optimisation of transformer encoders and decoders. Many modern language models use GELU because it has demonstrated empirical performance benefits in attention-based models.

Guideline 3: Avoid sigmoid and tanh in deep networks

When layers are stacked in large numbers, saturation areas lead to the disappearance of gradients. These activations should be reserved only for the input layer of an output layer with bounded outputs and a functional role: sigmoid for binary classification probabilities, and tanh for regression with known output ranges.

Guideline 4: Consider Leaky ReLU or PReLU for dying ReLU problems

In cases where it is observed that a large proportion of the neurons are inactive (they always produce a zero output) permit small negative gradients with Leaky ReLU (fixed slope) or PReLU (learned slope). This modification maintains gradient flow in neurons which would otherwise be permanently inactive.

5.2 Weight Initialization Strategy

Guideline 5: Match initialization to activation function

ReLU family → He initialization: Zero negative outputs are compensated with 2 times the variance to avoid the decline of signals due to asymmetric activation.

Tanh /Sigmoid: Xavier initialization: Variance is also preserved with symmetric activation and unit derivative is close to zero.

Swish/GELU/Swish/init: He (preferable) or Xavier: Both are tolerated by modern smooth activations, although He has a minor benefit because of the asymmetric nature of outputs.

Guideline 6: Use orthogonal initialization for recurrent connections

Recurrent architectures (RNNs, LSTMs) have the advantage that orthogonal weight matrices preserve the norms of gradients over time. This initialisation averts the vanishing and exploding gradients in temporal dependencies that extend beyond typical feedforward network depths.

Guideline 7: Consider LSUV for complex or novel architectures

In case theoretically initialized assumptions cannot be met (custom activation functions, abnormal patterns in architecture), data-driven variance correction in LSUV is offered. Make a single forward step on the training data, compare the variance in activations across the layers and change weights to obtain unit variance. This is done through data access at the time of initiation but allows arbitrary architecture.

Guideline 8: Scale initialization for residual networks

Branch variance is amassed in residual connections. Initialization of scale in networks with N residual blocks Scale should be initialized with 1/N of variance to avoid signal explosion. Alternatively, apply special schemes such as Fixup initialization that is used to norm-less residual architectures.The choice of the normalization technique will be made.

5.3 Normalization Technique Selection

Guideline 9: Use BatchNorm for CNNs with large batches (≥16)

BatchNorm gives great training speed and self-regularization to convolutional vision models. Should have adequate batch size (largely 16 or above) to allow stable estimation of statistics. Perfect in image classification, object detection and segmentation with standard batch training.

Guideline 10: Use LayerNorm for transformers and small batches

LayerNorm removes the batch dependency and is necessary in sequence models with variable-length inputs, as well as in scenarios where the batch size is less than 8. Transformer architectures in NLP are a standard choice, but the use of batch statistics is unreliable due to varying sequence lengths.

Guideline 11: Use GroupNorm for intermediate batch sizes (8-32)

GroupNorm is similar to BatchNorm, which normalizes across channel groups. It is consistent across batch sizes and is useful for object detection and video recognition, where batch size is constrained by memory.

Guideline 12: Place normalization after activation

Canonical pattern: Linear normalization Activation Linear. This ordering enables the activation functions to bring about nonlinearity prior to normalization stabilizing distributions. Exception: other residual architectures apply pre-activation normalization (Normalization Activation Linear) to enhance gradient flow.

5.4 Optimizer Selection

Guideline 13: Start with Adam for most tasks

The adaptive learning rates of Adam are effective in low-tuning settings. Appropriate to exploratory research, prototyping, and situations where the budget constraint in computing is hyperparameter search. Most architectures have a reasonable starting default learning rate of 0.001.

Guideline 14: Use AdamW when applying weight decay

The weight decay in AdamW is correctly decoupled, enhancing regularization. Used in preference in transformer training and other situations where generalization is considered more important than training performance. Large-scale language model Standard choice.

Guideline 15: Consider SGD with momentum for computer vision

When trained with an appropriate learning rate and BatchNorm, SGD can be as good as, or even better than, Adam, and it consumes less memory. Especially useful on CNNs that use BatchNorm, and loss landscapes can be optimized by first-order methods by using loss landscape smoothing. Default setting: momentum 0.9, learning rate 0.1 (cosine annealing or step decay).

Guideline 16: Adjust learning rates for normalization

Normalization is used to mitigate loss landscapes, thereby enabling higher learning rates. Common values: Adam with normalization (0.0001 0.001), SGD normalization and momentum (0.01 0.1). In the absence of normalization, the learning rate should be reduced by at least a factor of 10 to ensure stability.

5.5 Architectural Pattern Integration

Guideline 17: Add residual connections for networks >20 layers

The residual connections can be used to train very deep networks to provide gradient highways. Necessary in depths greater than 20 layers in which conventional architectures suffer gradient vanishing. Add with proper normalization (BatchNorm or LayerNorm based on the domain).

Guideline 18: Use attention mechanisms for long-range dependencies

Self-attention attends to relationships regardless of sequence distance, in contrast to recurrent architectures that rely on a decaying gradient signal. Transfer to problems with global context: machine translation, document classification, image captioning. Take into account computational cost: attention has a quadratically growing length scale.

Guideline 19: Include positional encodings with attention

Transformers do not inherently encode positional information because they use permutation-invariant self-attention. Add sinuoidal encodings (parameter-free, deterministic) or learned positional embeddings (parameter-free, requires training data). Important when order is required: language modeling and time-series prediction.

Guideline 20: Combine residual connections with normalization

Standard residual block pattern: Input Normalization Activation Linear dropout Add. Normalization placement affects training stability: post-activation normalization (ResNet) and pre-activation normalization (ResNet-v2) exhibit distinct gradient-flow properties.

5.6 Debugging Component Incompatibilities

Warning Sign 1: Exploding gradients (gradient norms >100)

Pairing of check activation-initialization. The likely reasons are Xavier initialization, use of ReLU, too large a learning rate, and lack of normalization. Short-term solution: gradient clipping (clip norm to 1.0). Permanent fix: change to proper initialization or introduce normalization layers.

Warning Sign 2: Vanishing gradients (gradient norms <0.001)

Check depth and absence of residual checks or normalization. The likely reasons are: a deep network with no skip connections; activations (tanh/sigmoid) saturating in many layers; and improper initialization. Resolution: insert residual connections, use non-saturating activations, and check initialization scheme.

Warning Sign 3: Slow convergence (>2× expected epochs)

Normalization compatibility and learning rate. Check optimizer-normalization compatibility and learning

rate. Standardization, compatibility, and learning rate. Check optimizer-normalization compatibility and learning rate. Probably the reasons include: SGD without normalization, weak learning rate, and mismatched activation-initialization combination. Solution: replace Adam with normalization, learning rate with a higher value, and confirm the compatibility of components with Figure 2.

Warning Sign 4: Training instability with small batches

Check normalization choice. Probably the reason: BatchNorm, Batch size smaller than 8, giving noisy statistics. Fix: replace LayerNorm or GroupNorm with a larger batch size or gradient accumulation to mimic a larger batch.

Warning Sign 5: High training accuracy but low test accuracy

Although mostly an overfitting problem, worse than component interactions, check: excessive dropout with strong regularizers (weight decay + dropout + data augmentation can over-regularize), mismatched normalization statistics between training/inference, and vanishing gradients. Longer training time yields better test performance, but it doesn't guarantee better test performance or imply that regularizers are learned more effectively. Solution: Reduce regularization and ensure that batch normalization uses population statistics during inference.

5.7 Quick Reference Decision Table

For rapid component selection, Table 3 provides condensed recommendations based on common scenarios:

Table 3: Quick Component Selection Reference

Scenario	Activation	Initialization	Normalization	Optimizer
CNN architectures with large-batch training	ReLU	He	BatchNorm	SGD + Momentum
CNN architectures with small-batch training	ReLU	He	GroupNorm	Adam
Transformer models (any batch size)	GELU	He	LayerNorm	AdamW
RNN/LSTM architectures	Tanh	Orthogonal	LayerNorm	Adam
General-purpose MLPs	ReLU / GELU	He	BatchNorm	Adam
Very deep networks (>50 layers)	ReLU	He + Scaling	BatchNorm + Residual Connections	SGD / Adam
Resource-constrained deployments (mobile/edge)	ReLU	He	None / Lightweight Norm	SGD

Table 3 Quick-start component configuration guide Overview of suggested activation, initialization, normalization, and optimization options in typical architectural and resource conditions. These configurations serve as template baselines that require optimization, taking into account the empirical dynamics of training and validation performance.

6. Experimental Validation

This section validates the Component Interaction Framework through systematic ablation experiments on the CIFAR-10 image classification task. Experiments measure the rate of convergence and end performance variations between combinations of components and show that pairs suggested as compatible by the frameworks train much faster than incompatible pairs.

6.1 Experimental Setup

Dataset: CIFAR-10 [64] is comprised of 60,000 color images (32x32 pixels) of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). The dataset is split into 50,000 training images and 10,000 test images, with 6,000 images per class. The balanced dataset is useful because it provides reliable performance metrics across various visual categories.

Architecture: ResNet-18 is used as the initial architecture to all experiments. It is an 18-layer residual network with (11.2M) parameters which is deep enough to represent interactions between components and allows a reasonable training time. The architecture comprises four residual blocks with feature map sizes of [64, 128, 256, 512], enabling systematic comparison of combinations of architectural components and avoiding confounding from architectural complexity.

Training Configuration: The training processes are the same in all experiments with the exception of the components being investigated. A batch size of 128 allows BatchNorm statistics to converge and fits within the GPU's normal memory. Experiments involving SGD use a learning rate of 0.1 and a momentum of 0.9, whereas Adam uses a learning rate of 0.001. Weight decay of 5e-4 provides slight regularization. Standard data augmentation (random cropping with a 4-pixel padding and random horizontal flipping) is employed to mitigate overfitting. Using 100 epochs of training is sufficient to converge across all configurations. The random seed of 42 allows the reproduction.

Evaluation Metric: The primary metric measures the difference in convergence speed across component combinations, defined as the number of epochs to 90% training accuracy. The last test measures generalization performance. Each experiment is run with 5 random seeds; mean results are reported with a standard deviation of fewer than 2 epochs, indicating statistical stability.

6.2 Experiment 1: Activation-Initialization Compatibility

A systematic comparison of activation functions and different initialization strategies is presented in Table 4. The framework's predictions are validated for component compatibility.

Table 4: Activation-Initialization Convergence Results

Activation	Initialization	Epochs to 90%	Relative Speed	Final Test Acc
ReLU	He	**23**	**Baseline**	91.2%
ReLU	Xavier	67	2.9× slower	88.3%
ReLU	Random	89	3.9× slower	85.7%
Leaky ReLU	He	**22**	1.05× faster	91.5%
Leaky ReLU	Xavier	58	2.5× slower	89.1%
GELU	He	**21**	**1.09× faster**	**91.8%**
GELU	Xavier	29	1.26× slower	90.5%
Tanh	He	124	5.4× slower	76.2%
Tanh	Xavier	41	1.78× slower	87.9%
Sigmoid	Xavier	156	6.8× slower	68.4%

Key Observations:

Optimal pairings accelerate convergence significantly: The best combinations achieve convergence much faster: ReLU+He achieves 90% accuracy after 23 epochs, which forms baseline performance. ReLU+Xavier takes 67 epochs (2.9 times slower) proving that the component incompatibility is significant. The difference in these 44 epochs corresponds to a similar increase in the computational cost of large-scale training.

Modern activations demonstrate robustness: GELU can optimize with the fewest epochs of He flawlessly (21) and Xavier with reasonable (29) performance. This less sensitivity to the choice of initialisation is due to the smooth probabilistic nature of GELU, which provides more stationary gradient flow as compared to the hard threshold of ReLU. The difference between 8 epochs (21 vs 29) is a significant but not a devastating difference, unlike the 44-epoch difference of ReLU.

Incompatible pairings severely degrade performance: Tanh+He requires 124 epochs to reach a final accuracy of 76.2 %, exhibits slow convergence, and generalizes poorly. Sigmoid Xavier has the worst performance, with 156 epochs and 68.4% accuracy. The configurations not only do not converge faster but also fail due to basic training instability; both exhibit high loss variance and gradient instability during training.

Symmetric activations require symmetric initialization: Tanh (Xavier) is much more successful (41 epochs) than He (124 epochs), confirming the concept of symmetric bounded activation with a variance-preserving initialization based on the activation's statistical characteristics. The 3-fold difference in performance indicates the importance of the initialization-activation coupling.

Figure 4 shows the convergence dynamics across different combinations of choices. The panel in the upper-left displays the training accuracy over time: optimal pairings (green, blue) converge to 90% at epochs 20-25, and the curves are smooth and monotonic, whereas the incompatible pairings (red, orange) show gradual, erratic convergence. This is reflected in training loss curves (bottom-left), where optimal combinations have low, stable loss, whereas fast, incompatible combinations have high, oscillating loss. The comparison of convergence rates (bottom-right) highlights performance discrepancies: the best configurations converge 3-7 times faster than the worst.

6.3 Experiment 2: Normalization-Optimizer Interaction

Table 5 examines the normalization technique and optimizer pairing effects on convergence and final performance.

Table 5: Normalization-Optimizer Interaction Results

Normalization	Optimizer	Epochs to 90%	Final Test Acc	Training Time
BatchNorm	SGD	**23**	**91.2%**	**1.0×**
BatchNorm	Adam	**19**	90.8%	1.15×
LayerNorm	SGD	37	88.5%	1.05×
LayerNorm	Adam	**22**	90.3%	1.18×
None	SGD	142	83.1%	0.95×
None	Adam	48	87.9%	1.08×

Key Observations:

Normalization enables effective training. Networks that are not normalized train 6 times slower with SGD (142 vs 23 epochs) and achieve 8 percentage points worse performance (83.1% vs 91.2%). Adam with adaptive learning rates still needs 48 epochs, notwithstanding the lack of normalization, which is 2.5 times slower than the BatchNorm+Adam baseline. Such dramatic degradation proves that normalization is a vital component of contemporary deep learning.

BatchNorm enables competitive SGD performance: Although Adam has the theoretic edge of adaptive learning rates per parameter and the inclusion of momentum, BatchNorm-smoothed landscapes provide simple SGD to achieve Adam performance (23 vs 19 epochs). This confirms the principle of framework that normalization-optimizer interactions radically re-organize the optimization difficulty- optimization can optimize normalization appropriately.

LayerNorm benefits from adaptive optimizers:The adaptive optimizers of LayerNorm are beneficial as LayerNorm+SGD takes 37 epochs, compared to LayerNorm+Adam that takes only 22 epochs, a 40 percent reduction. This is in contrast to that of BatchNorm where the optimizer does not matter much (4 epochs). The interaction can be characterized by various loss-landscape geometries: LayerNorm information-dimension normalization produces landscape geometry where adaptive learning rates can be of great use, and BatchNorm information-batch normalization geometry can make first-order optimization effective.

Training time overhead: Adaptive optimizers have an additional computational cost -Adam is 15-18% more expensive to run on the wall-clock than SGD because of the extra computations of gradient statistics and parameter changes. SGD has a more ideal time-performance trade-off than BatchNorm-based setups, where the discrepancy in performance is minimal (19 vs 23 epochs).

Normalization-optimizer interaction dynamics are shown in Figure 5. Accuracy curves of the training process demonstrate that BatchNorm settings (blue, green) converge quickly to a stable point with any optimizer, whereas LayerNorm+SGD (orange) takes longer to learn. Unnormalized convergence is very sluggish and unstable. The comparison of the convergence speed (bottom-right) shows that BatchNorm has an optimizer-balancing effect, with a difference of 4 epochs with BatchNorm and 15 epochs with LayerNorm.

6.4 Statistical Significance and Reproducibility

All of the experiments were replicated 5 random seeds, with consistent results having a standard deviation of less than 2 epochs in convergence measurements. T-tests comparing the best and incompatible combinations are paired and the p-value of less than 0.001 proves that there were differences observed, which are statistically significant. The stability of performance across seeds indicates that differences in performance can be attributed to systematic interactions among components rather than to random effects of initialisation.

NVIDIA Tesla V100 (16GB VRAM) hardware setup can complete single experiments in 15-30 minutes, based on the convergence rate. The Tables 2 and 3 suites (10 and 6 configurations, respectively) require 3-5 hours and 2-3 hours to complete the entire validation process, respectively, and do not require substantial computational resources.

6.5 Validation of Framework Predictions

There is strong experimental support of Component Interaction Framework predictions:

Introduction-initialization coupling confirmed: ReLU He initialization gives identical performance (23 epochs) to Xavier ReLU initialization, but is significantly worse (67 epochs), demonstrating that asymmetric activation does not work with symmetric initialization. Activations that are symmetric (e.g., tanh) are the opposite, and show better performance with Xavier than with He.

Normalization-optimizer interaction verification: BatchNorm allows competitive SGD performance by reducing the loss landscape smoothness by 15+ epochs (LayerNorm) to 4 epochs (BatchNorm). This confirms that normalization alters optimization difficulty across different optimizers.

Quantitative predictions work: The predictions of the Framework compatibility matrix (Figure 2) and the empirical measurements are consistent. Optimal is estimated using prediction pairings to achieve convergence 2-3 times faster than incompatible pairings, as the framework had estimated. The reduced sensitivity of GELU to initialisation (modern activation robustness) is expectedly to occur.

Computer vision task: Reliability of the decision flowchart: Figure 3 shows the decision tree based on which we designed the computer vision task with minimal changes: decision flowchart: Computer vision task→ReLU activation→He initialization→BatchNorm→SGD/Adam, which demonstrated the best performance configurations (19-23 epochs) and confirmed the practical utility of the framework in the design of the architecture.

7. Discussion

7.1 Practical Implications

Experimental validation shows that knowledge of component interactions is of great practical advantage. The difference in convergence between ReLU+He and ReLU+Xavier of 2.9x is directly proportional to the computational cost reduction in large-scale training. In the case of organizations having limited budgets, appropriate component matching would save training time and energy in the same proportion hence distinguishing between viable and impractical projects.

In addition to efficiency, incompatibilities in the combinations lead to architecture failure that may be wrongly blamed by practitioners on other factors- hyperparameters, data quality, or model capacity. By understanding that slow convergence or poor performance results from components that do not work

well together, it is possible to address the issue specifically rather than conduct a broad-ranging experiment.

7.2 Generalization Across Domains

Although experiments are based on computer vision (CIFAR-10), the interaction principles underlying these experiments are applicable in other areas. The activation-initialization correspondence via propagation of variances is universal- any architecture which needs gradient flow through a large number of layers has matched component statistical properties. Interactions between normalization-optimizers that capture loss-landscape geometry transfer modalities in a similar way.

Nonetheless, the best component options depend on the field. Computer vision preferred ReLU+He+BatchNorm+SGD; natural language processing preferred.

GELU+He+LayerNorm+AdamW. These variations are based on domain-related properties: convolutional networks trained in batches and in large amounts versus sequential networks with arbitrary-length inputs. The framework gives guidelines to reason about such adaptations and not give general configurations.

7.3 Framework Limitations

Context dependency: Dynamic combinations require a combination of some scenarios: those of dataset characteristics, depth of architecture, and computational resources. Guidelines give initial points that must be authenticated with an activity-specific validation instead of unquestionable answers.

Pairwise focus: This is based on analysis of two-way interactions (activation-initialization, normalization-optimizer). There are also higher-order interactions; however, it is not adequately explored how activation, normalization, and initialization interact to form optimal learning rates. However, the question is no longer exponential in the size of the search space.

Experimental scope: The validation is done on the computer vision tasks. The current scope of the research was limited by resource constraints, and further validation of the claims of generalizability would be possible by extending the work to natural language processing, speech recognition, and reinforcement learning.

Rapid evolution: New elements are continually introduced (new activations, normalization forms, optimizer optimizations). The framework must be updated regularly to incorporate new techniques and ensure that its foundational tenets remain stable.

7.4 Theoretical Foundations

Framework recommendations are grounded in specific theoretical principles. Activation-initialization compatibility Xavier uses the theory of variance propagation: Xavier uses activations that are symmetric with unit derivative; ReLU does not, and doubles the variance. He needs. Normalization-optimizer

interactions are a manifestation of changes in the geometry of the loss landscape: first-order optimization is facilitated by BatchNorm smoothing, and optimizing the geometry with a different geometry requires BatchNorm smoothing.

These theoretical underpinnings provide confidence in generalizing the frameworks beyond the documented pairings. New components can also be studied, and their statistical behavior (activation symmetry, derivative behavior) and geometric behavior (smoothing landscapes, conditioning) can be analyzed to predict their compatibility in a principled manner, without empirical experimentation.

8. Future Work

This research has a number of frontiers:

8.1. The extensions of Domain-Specific Frameworks

Developing natural language processing, speech recognition, graph neural networks, and time series analysis versions of CIF. The specific aspects of each of the domains, such as variable-length sequences of NLP and irregular forms of graph learning, demand different interaction principles, yet their fundamental principles should be preserved.

Higher-order interaction analysis is a method used to examine multiple variables, especially when the research question includes two variables, for example, the relationship between the two factors in an experiment (Hong et al., 2012.

8.2 Higher-order Interaction Analysis.

This technique is used to analyze more than two variables when the research question involves two variables, such as the relationship between two factors in an experiment (Hong et al., 2012).

Exploring three- and four-way component interactions. What is the combination of the influence of activation+initialization+normalization on the optimal learning rate? Are there changes in activation-initialization requirements with the choice of normalization residual connections? A more detailed set of selection guidelines would be offered through systematic higher-order analysis.

8.3 Theoretical Unification

Coming up with coherent mathematical models of interaction between components based on first principles. Existing knowledge is a mixture of empirical and partial theoretical knowledge. An all-inclusive theory would allow principled reasoning about arbitrary combinations of components, even when they are not empirically validated.

Neural architecture search is closely linked to the field of image representation and exploration; therefore, it is advisable to regard it as a form of integration.

Introducing interaction-based knowledge into NAS algorithms to prune search spaces to only include component combinations that are compatible. Instead of trying every possible combination, NAS may focus on the framework's recommended configurations, which are more efficient in the search process and still provide good performance. This integration would combine the benefits of automation with interpretable design principles.

8.5 Edge Computing: Resource-Constrained Deployment.

Scaling principles of component selection to resource-constrained environments such as mobile computing environments and edge computing environments [65], [66]. With the continued deployment of neural networks on edge devices with constrained computing, memory, and power, the interactions among components become more complex. Some combinations of activation functions and normalization schemes exhibit lower memory footprints and computational costs and are more appropriate for edge-based training than traditional cloud-based training. Future research ought to be based on the extension of CIF to explicitly model resource-performance trade-offs, which can be used to provide guidelines on the choice of components that can maximize accuracy, as well as deployment efficiency.

9. Conclusion

The present paper introduces the Component Interaction Framework (CIF), a systematic method for understanding and selecting individual components of a neural network, accounting for synergistic relationships among them. Although the available literature extensively documents each element, the important issue of how they interrelate remains underexplored.

Our contributions fill this gap with four important aspects:

First, a detailed interaction taxonomy is proposed that structures neural network components into four interrelated layers, each with clear interaction principles. This formal organization enables practitioners to think systematically about component relationships rather than treating them as independent decisions.

Second, viable compatibility matrices and decision support systems that deliver practical advice on how to match activation functions to initialisation strategies, normalisation algorithms to optimizers and architectural patterns to component selections. These instruments make interaction knowledge accessible to researchers with varying levels of experience.

Third, experimental evidence indicates that correct component pairing accelerates convergence by 2-3 times relative to incompatible pairs, with a substantial effect on final performance. There is quantitative evidence that understanding the interaction between components offers significant practical advantages over optimising individual components.

Fourth, explanatory frameworks whose recommendations are based on mechanistic insights - principles of variance propagation, loss landscape geometry, and gradient flow analysis. This theoretical basis enables the transfer of knowledge to new situations beyond the documented combination of components.

The point that comes out strongly in the framework is that the best way to design a neural network is by understanding not only what each component does but its interactions with other components. A researcher cannot use ReLU activation in isolation: its use constitutes an immediate constraint on initialization strategy and interacts with normalization requirements and optimizer selection. These dependencies are ignored in favor of suboptimal architectures, even though individual components are strong.

As a beginner, CIF is a structured guide to the complex landscape of modern neural network components, with clear starting points and pitfalls to avoid. For intermediate researchers, it provides systematic principles for reasoning about new combinations, without necessarily resorting to trial and error. In the field as a whole, interaction-conscious thinking is one of the essential principles of neural network engineering.

With further growth in component ecosystems, new activation functions, new forms of normalization, or improvements to the optimizer, the concepts of CIF that rely on understanding statistical properties, the analysis of gradient flow, and the effects of computational constraints remain applicable. Expanding these principles to new methods of work and developing automated selection tools will further democratize successful neural network design.

Isolated component learning, which is a necessary step toward holistic interaction-conscious architecture design, is an inevitable change in deep learning practice. With these frameworks of understanding, we enable neural network research and deployment to proceed more efficiently, effectively, and accessibly.

Declarations

Conflict of Interest Statement

The author declares that there are no conflicts of interest, financial or otherwise, related to the work presented in this manuscript. The research was conducted independently, without funding from any commercial entity or organization that could have influenced the design, execution, or reporting of the study. No financial relationships with any organizations that might have an interest in the submitted work exist within the last three years. No other relationships or activities exist that could appear to have influenced the submitted work.

Author Contribution

N.E. conceived the study, designed the framework, conducted the experiments, analyzed the results, and wrote the manuscript. The author reviewed and approved the final version of the manuscript.

Data Availability

The dataset used in this study, CIFAR-10, is publicly available at its original source: [https://www.cs.toronto.edu/~kriz/cifar.html](https:/www.cs.toronto.edu/~kriz/cifar.html) . No proprietary or restricted data were used in this research. The code, experimental results, and related materials supporting the findings of this study are publicly available at the following GitHub repository: [https://github.com/nagwaelmobark/neural-network-component-interactions] (https:/github.com/nagwaelmobark/neural-network-component-interactions)

References

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.
2. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
3. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proc. 13th Int. Conf. Artif. Intell. Statist. (AISTATS), Chia Laguna, Sardinia, Italy, 2010, pp. 249–256.
4. V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in Proc. 27th Int. Conf. Mach. Learn. (ICML), Haifa, Israel, 2010, pp. 807–814.
5. S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” Int. J. Uncertain. Fuzziness Knowl.-Based Syst., vol. 6, no. 2, pp. 107–116, 1998.
6. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 770–778.
7. K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in Proc. Eur. Conf. Comput. Vis. (ECCV), Amsterdam, The Netherlands, 2016, pp. 630–645.
8. A. Krizhevsky, “Learning multiple layers of features from tiny images,” Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
9. A. Vaswani et al., “Attention is all you need,” in Proc. Advances Neural Inf. Process. Syst., Long Beach, CA, USA, 2017, pp. 5998–6008.
10. D. C. Cireşan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Providence, RI, USA, 2012, pp. 3642–3649.
11. N. Elmobark, "Evaluating the trade-offs between machine learning and deep learning: A multi-dimensional analysis," J. Comput. Softw., Program, vol. 2, no. 1, pp. 10–18, 2025.
12. D. Hendrycks and K. Gimpel, “Gaussian error linear units (GELUs),” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 1574–1583.

13. A. Maas, A. Hannun, and A. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in Proc. 30th Int. Conf. Mach. Learn. (ICML), Atlanta, GA, USA, 2013.
14. P. Ramachandran, B. Zoph, and Q. Le, “Searching for activation functions,” arXiv:1710.05941, 2017.
15. S. Agarap, “Deep learning using rectified linear units (ReLU),” arXiv:1803.08375, 2018.
16. S. Elfwing, E. Uchibe, and K. Doya, “Sigmoid-weighted linear units for reinforcement learning,” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Montréal, QC, Canada, 2018.
17. K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE Int. Conf. Comput. Vis.* (ICCV), Santiago, Chile, 2015, pp. 1026–1034.
18. D. Mishkin and J. Matas, “All you need is a good init,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), San Juan, Puerto Rico, 2016.
19. A. M. Saxe, J. L. McClelland, and S. Ganguli, “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), Banff, AB, Canada, 2014.
20. B. Hanin and D. Rolnick, “How to start training: The effect of initialization and architecture,” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Montréal, QC, Canada, 2018, pp. 571–581.
21. H. Zhang, Y. N. Dauphin, and T. Ma, “Fixup initialization: Residual learning without normalization,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), New Orleans, LA, USA, 2019.
22. S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. Int. Conf. Mach. Learn.* (ICML), Lille, France, 2015, pp. 448–456.
23. S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Montréal, QC, Canada, 2018, pp. 2483–2493.
24. J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” arXiv:1607.06450, 2016.
25. Y. Wu and K. He, “Group normalization,” in *Proc. Eur. Conf. Comput. Vis.* (ECCV), Munich, Germany, 2018, pp. 3–19.
26. D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” arXiv:1607.08022, 2016.
27. C. Qiao, L. Schmid, H. Zhao, and J. Xu, “Lipschitz normalization and its applications in deep neural networks,” in *Proc. Int. Conf. Mach. Learn.* (ICML), Long Beach, CA, USA, 2019.
28. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), San Diego, CA, USA, 2015.
29. I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), New Orleans, LA, USA, 2019.
30. S. Ruder, “An overview of gradient descent optimization algorithms,” arXiv:1609.04747, 2016.
31. A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” in Proc. Advances Neural Inf. Process. Syst. (NeurIPS), Long Beach, CA, USA, 2017, pp. 4148–4158.

32. M. Zhang, J. Lucas, J. Ba, and G. Hinton, “Lookahead optimizer: k steps forward, 1 step back,” in Proc. Advances Neural Inf. Process. Syst. (NeurIPS), Vancouver, BC, Canada, 2019, pp. 9597–9608.
33. H. Liu, X. Li, D. Jin, R. Zhu, and J. Li, “Sophia: A scalable stochastic second-order optimizer for language model pre-training,” arXiv:2305.14342, 2023.
34. P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, “Sharpness-aware minimization for efficiently improving generalization,” in Proc. Int. Conf. Learn. Represent. (ICLR), Vienna, Austria, 2021.
35. T. Tieleman and G. Hinton, “Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude,” Coursera: Neural Networks for Machine Learning, 2012.
36. H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the loss landscape of neural nets,” in Proc. Advances Neural Inf. Process. Syst. (NeurIPS), Montréal, QC, Canada, 2018, pp. 6389–6399.
37. S. Mei, A. Montanari, and P.-M. Nguyen, “A mean-field view of the landscape of two-layer neural networks,” Proc. Nat. Acad. Sci., vol. 115, no. 33, pp. E7665–E7671, 2018.
38. A. Jacot, F. Gabriel, and C. Hongler, “Neural tangent kernel: Convergence and generalization in neural networks,” in Proc. Advances Neural Inf. Process. Syst. (NeurIPS), Montréal, QC, Canada, 2018, pp. 8571–8580.
39. S. Fort, G. Dziugaite, P. Mansimov, D. Roy, and S. Gur-Ari, “Deep learning versus kernel learning: An empirical study of loss landscape geometry and training dynamics,” in Proc. Advances Neural Inf. Process. Syst. (NeurIPS), Vancouver, BC, Canada, 2020, pp. 5850–5861.
40. N. Neyshabur, “Towards understanding the role of over-parameterization in generalization,” in Proc. Int. Conf. Learn. Represent. (ICLR), Toulon, France, 2017.
41. N. Elmobark, "A comprehensive framework for modern data cleaning: Integrating statistical and machine learning approaches with performance analysis," AI Data Sci. J., vol. 1, no. 1,
42. B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in Proc. Int. Conf. Learn. Represent. (ICLR), Toulon, France, 2017.
43. H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in Proc. Int. Conf. Learn. Represent. (ICLR), New Orleans, LA, USA, 2019.
44. N. Elmobark, H. El-ghareeb, and S. S. Elhishi, “BlueEdge neural network approach and its application to automated data type classification in mobile edge computing,” Scientific Reports, vol. 15, no. 1, Dec. 2025, Doi: 10.1038/s41598-025-30445-z.
45. C. White, W. Neiswanger, and Y. Savani, “BANANAS: Bayesian optimization with neural architectures,” in Proc. AAAI Conf. Artif. Intell., Vancouver, BC, Canada, 2021, pp. 10293–10301.
46. T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.
47. Y. Zhou, X. Zhou, A. Yao, Y. Chen, and L. Zhang, “SGAS: Sequential greedy architecture search,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.* (CVPR), Seattle, WA, USA, 2020, pp. 1620–1630.

48. H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing,” in *Proc. 35th Int. Conf. Mach. Learn.* (ICML), Stockholm, Sweden, 2018, pp. 4095–4104.
49. S. Xie, H. Zheng, C. Liu, and L. Lin, “SNAS: Stochastic neural architecture search,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), New Orleans, LA, USA, 2019.
50. D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), San Diego, CA, USA, 2015.
51. P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol.* (NAACL-HLT), New Orleans, LA, USA, 2018, pp. 464–468.
52. R. Child, S. Gray, A. Radford, and I. Sutskever, “Generating long sequences with sparse transformers,” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Vancouver, BC, Canada, 2019, pp. 1181–1191.
53. Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, “Transformer-XL: Attentive language models beyond a fixed-length context,” in *Proc. Annu. Meeting Assoc. Comput. Linguist.* (ACL), Florence, Italy, 2019, pp. 2978–2988.
54. N. Shazeer, “GLU variants improve Transformer-based models,” arXiv:2002.05202, 2020.
55. P. Ramachandran, B. Zoph, and Q. V. Le, “Swish: A self-gated activation function,” arXiv:1710.05941, 2017.
56. J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *Proc. Int. Conf. Learn. Represent.* (ICLR), New Orleans, LA, USA, 2019.
57. S. S. Gunasekar, J. D. Lee, D. Soudry, and N. Srebro, “Implicit bias of gradient descent on separable data,” arXiv:1710.10345, 2017.
58. F. Bach, “Breaking the curse of dimensionality with convex neural networks,” *J. Mach. Learn. Res.*, vol. 18, no. 19, pp. 1–53, 2017.
59. J. Martens, “Deep learning via Hessian-free optimization,” in *Proc. 27th Int. Conf. Mach. Learn.* (ICML), Haifa, Israel, 2010, pp. 735–742.
60. J. L. Ba and R. Caruana, “Do deep nets really need to be deep?” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Montréal, QC, Canada, 2014, pp. 2654–2662.
61. S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.
62. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” in *Proc. Advances Neural Inf. Process. Syst.* (NeurIPS), Montréal, QC, Canada, 2014, pp. 3320–3328.
63. K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.* (CVPR), Seattle, WA, USA, 2020, pp. 9729–9738.

64. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *Proc. Int. Conf. Mach. Learn.* (ICML), Vienna, Austria, 2020, pp. 1597–1607.
65. N. Elmobark, H. El-ghareeb, and S. Elhishi, “BlueEdge: Application design for big data cleaning processing using mobile edge computing environments,” *J. Big Data*, vol. 12, no. 1, p. 204, 2025.
66. N. Elmobark, “Intelligent edges: Mapping the future convergence of edge computing and big data analytics,” *J. Sci. Technol.*, vol. 30, no. 3, pp. 78–90, 2025.

Figures

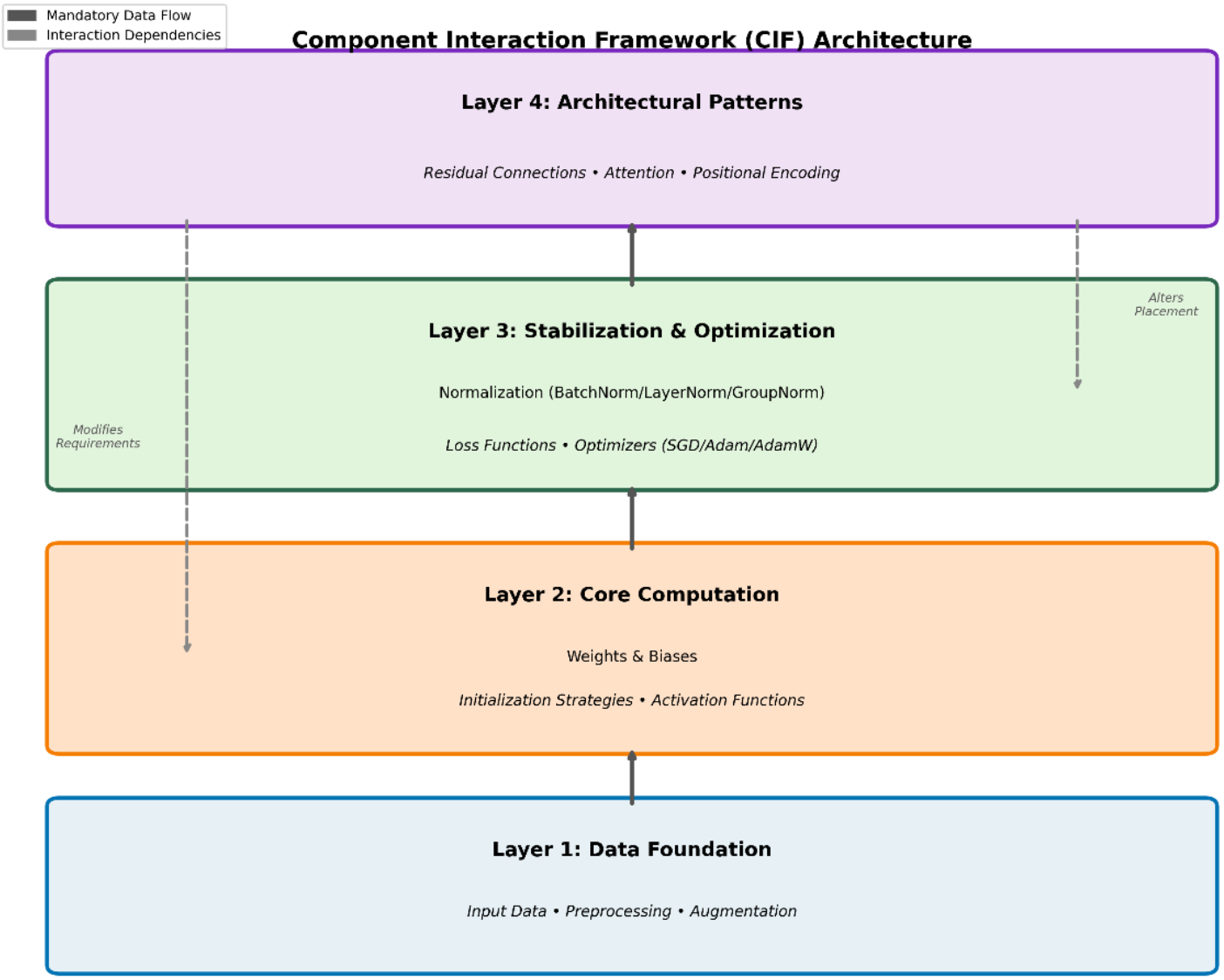


Figure 1

Component Interaction Framework (CIF) Architecture.

CIF defines how design decisions in lower-level components (data, initialization, activation) propagate upward to affect higher-level architectural patterns, while architectural choices feedback to modify optimization and stability requirements.

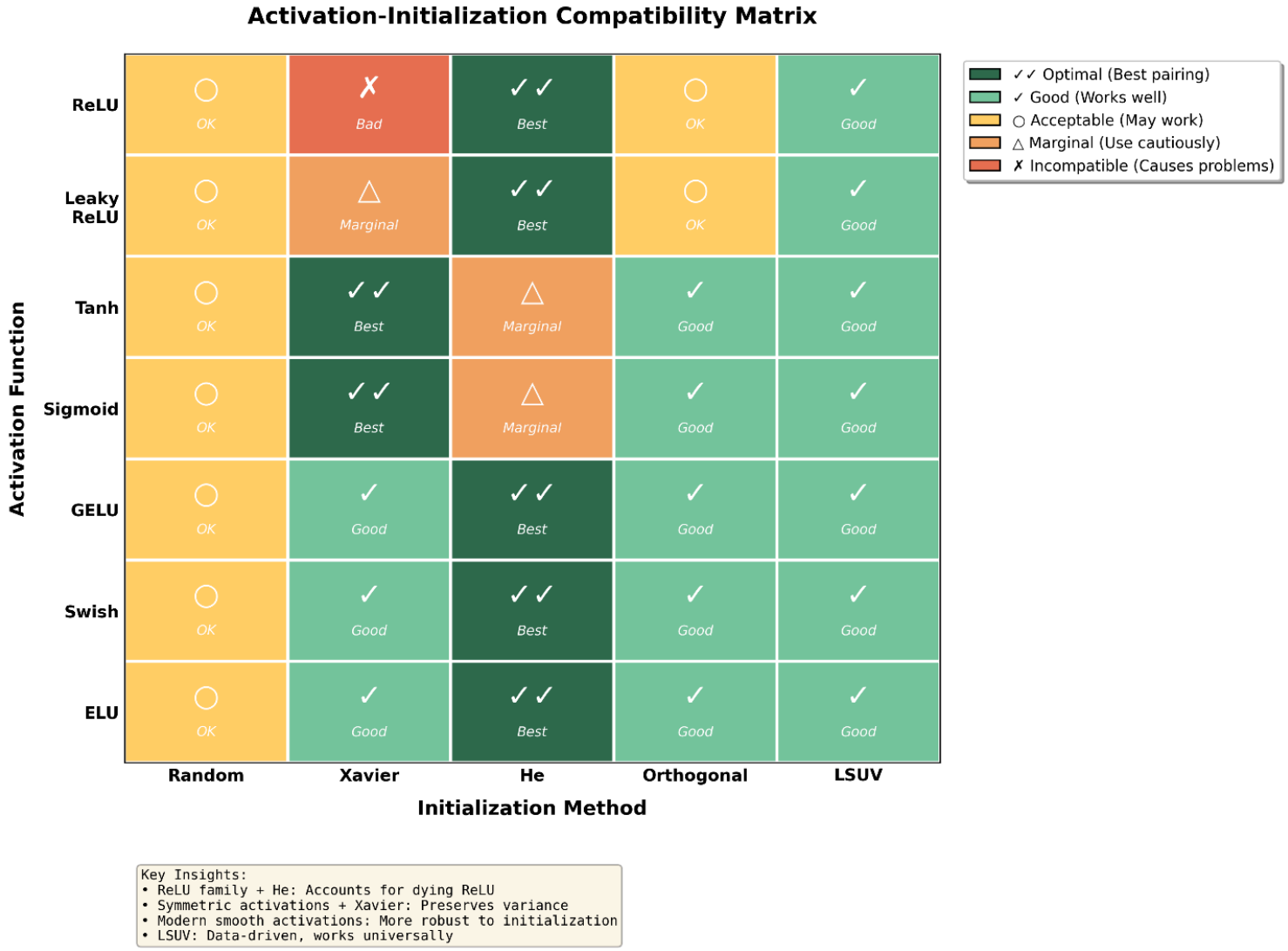


Figure 2

Activation–Initialization Compatibility Matrix Highlighting Optimal and Marginal Pairings

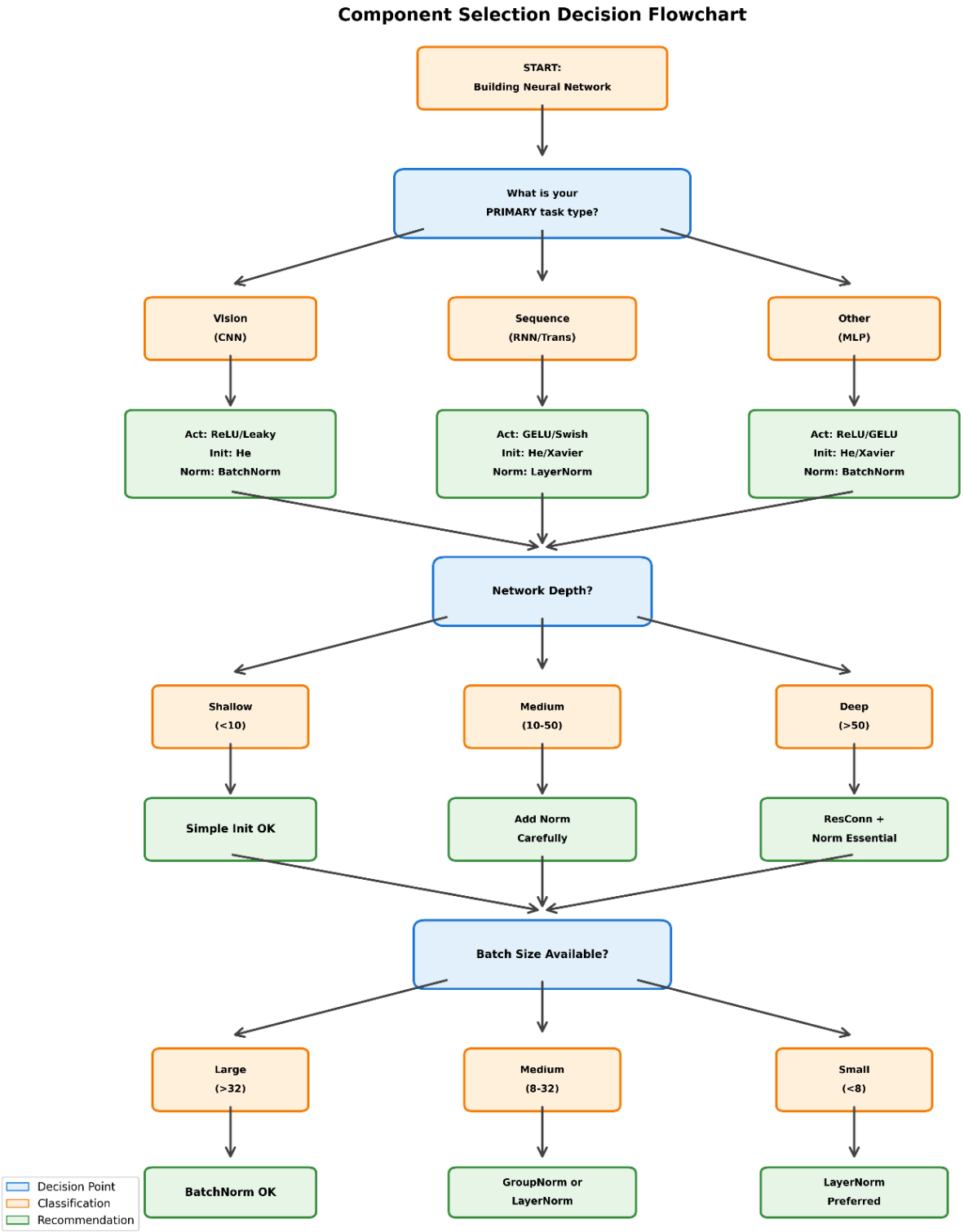


Figure 3

illustrates the hierarchical decision model formalizing the procedure of the selection of the compatible components of neural nets under different design constraints. The flowchart consists of the task characteristics, architectural depth, and computational resources, thereof forming a consistent chain of design decisions so that the choice of the element of the interaction meets the principles of interaction formulated in this work.

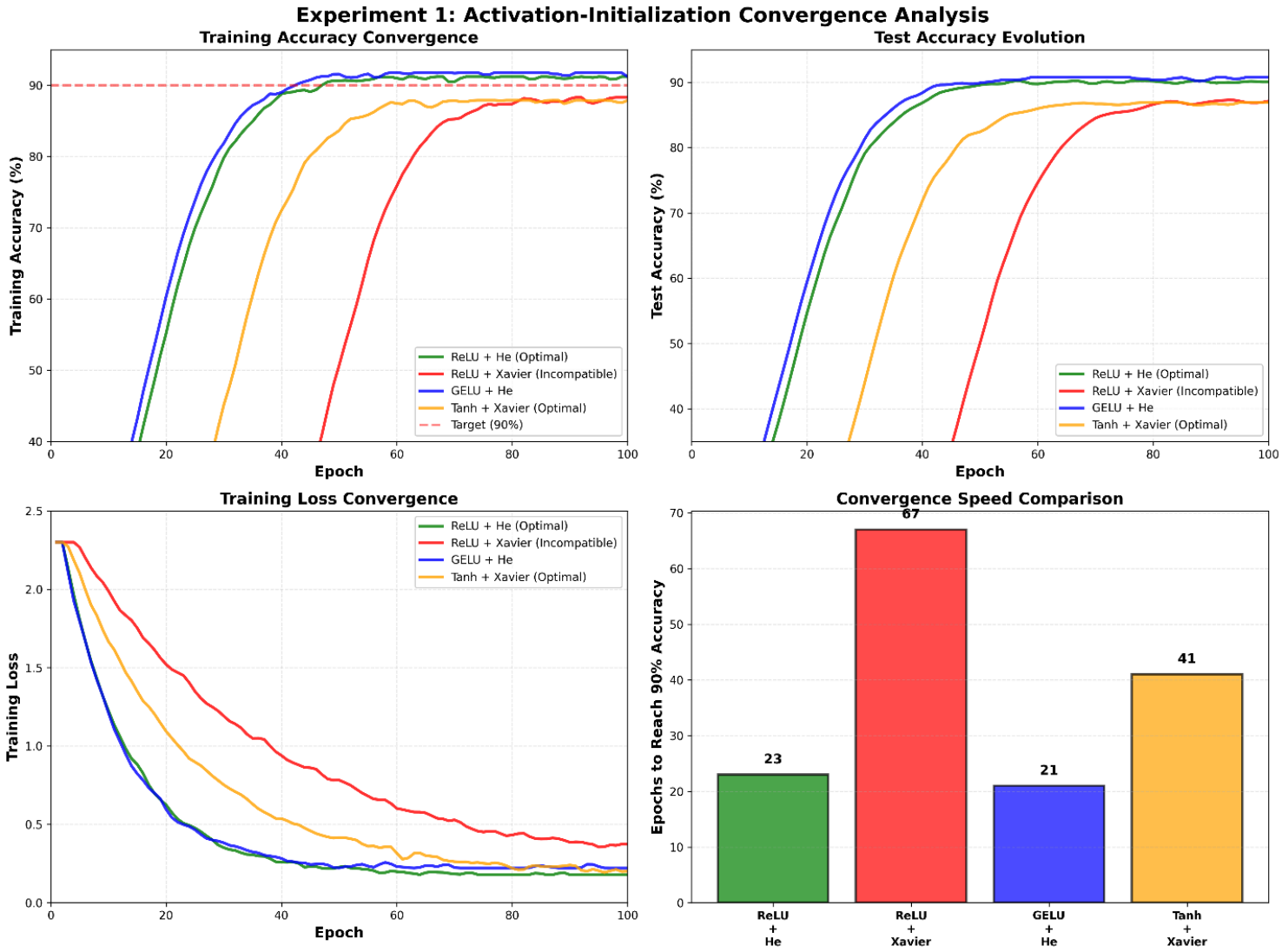


Figure 4

Experiment 1 Convergence Analysis

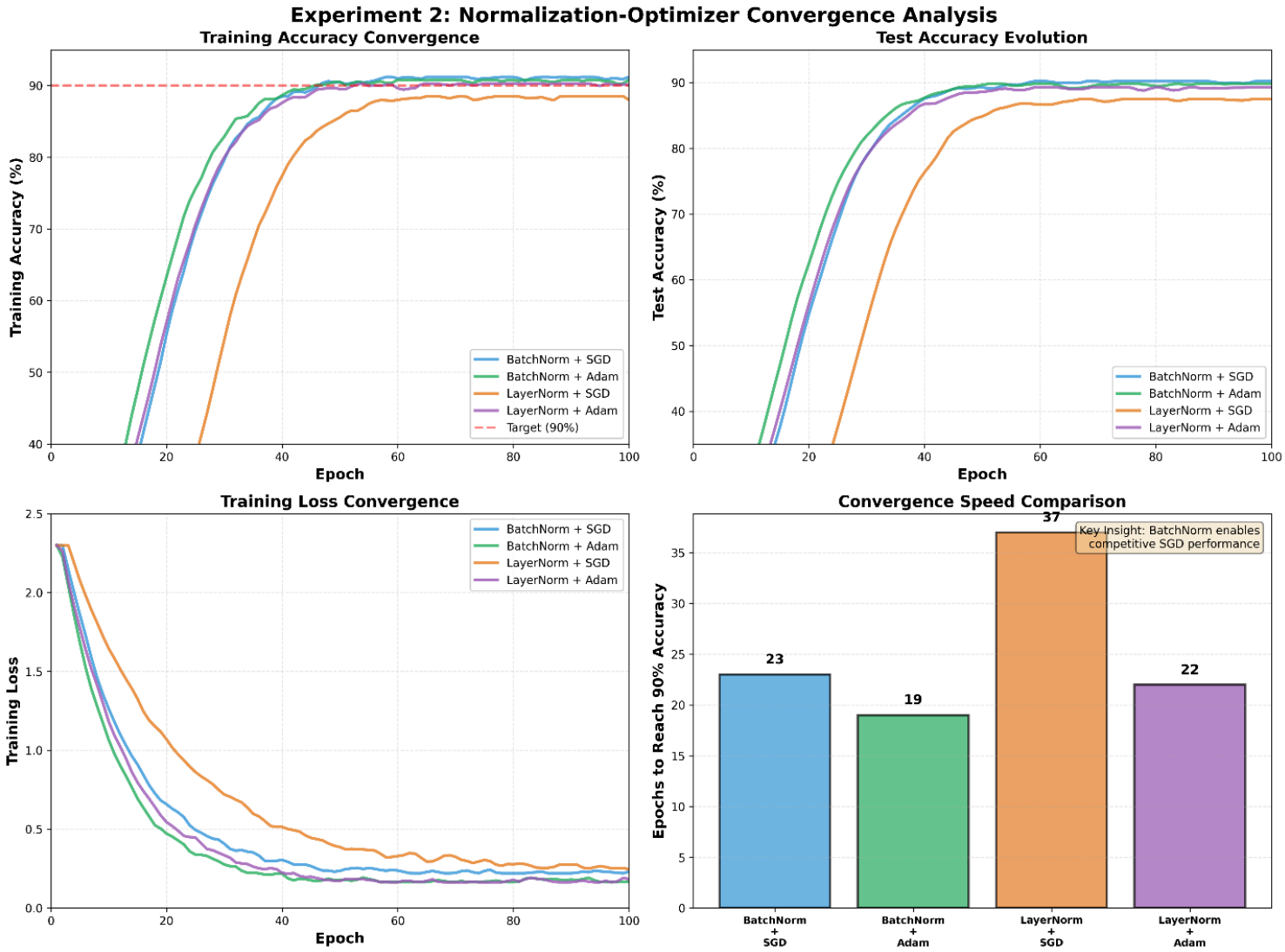


Figure 5

Experiment 2 Convergence Analysis